helion.pl

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Geometria obliczeniowa. Wprowadzenie

Autorzy: Franco P. Preparata, Michael Ian Shamos
Tłumaczenie: Tomasz Żmijewski
ISBN: 83-7361-098-7
Tytuł oryginału: Computational Geometry. An Introduction
Format: B5, stron: 386

W ostatniej dekadzie systematyczne badania algorytmów geometrycznych spowodowały utworzenie nowej dziedziny badawczej – geometrii obliczeniowej. Jej osiągnięcia mają szerokie zastosowanie w przeżywającej ostatnio błyskawiczny rozwój trójwymiarowej grafice komputerowej, a także w automatyce, robotyce i w statystyce. Książka niniejsza to obszerny, systematyczny i jednolity wykład na ten temat. Stanowi ona klasyczną pozycję w tym zakresie informatyki.

Najważniejszym zadaniem geometrii obliczeniowej jest wskazanie pojęć, właściwości i technik, które będą pomocne przy tworzeniu sprawnych algorytmów rozwiązujących problemy z dziedziny geometrii.

Tematy poruszane w tej książce, to między innymi:

- podstawy geometrii i historia geometrii obliczeniowej
- wyszukiwanie geometryczne
- uzyskiwanie informacji o obiektach
- tworzenie otoczki wypukłej wraz z szeregiem problemów z tym zagadnieniem związanych,
- sąsiedztwo, przecięcia oraz geometria prostokątów

W książce metody geometrii obliczeniowej prezentowane są przez szczegółowe omówienie konkretnych przypadków. Początkowo książka ta miała być podręcznikiem dla studentów, ale w jej obecnym kształcie będzie przydatna także dla badaczy i dla osób zawodowo zajmujących się projektowaniem wspomaganym komputerowo, grafiką komputerową i robotyką.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

1
Wprowadzenie

1.1. Rys historyczny

Geometria starożytnych Egipcjan i Greków to dzieła sztuki matematyki stosowanej. Problemy geometryczne pojawiały się w związku z koniecznością dokładnego wyliczania różnych obciążeń o charakterze podatkowym czy podczas wznoszenia monumentalnych budowli. Jak to nieraz w historii bywa, matematyka — stworzona jako narzędzie przydatne faraonom w rządzeniu państwem — wyrosła znacznie ponad pierwotnie stawiane przed nią zadania, a geometria stała się szkołą myślenia matematycznego. To w geometrii tak przydatna jest zwykła intuicja, a nowe odkrycia są w zasięgu nawet amatorów.

Powszechnie uważa się, że najważniejszym wkładem Euklidesa w rozwój geometrii jest przedstawienie metody aksjomatycznego przeprowadzania dowodów. Nie będziemy tutaj z tą tezą dyskutować. Dla nas jednak znacznie ważniejsze będzie pojęcie *konstrukcji euklidesowej*: metody obejmującej algorytm konstrukcji *wraz z dowodem*. Konstrukcje euklidesowe spełniają wszystkie wymogi stawiane przed algorytmami: są jednoznaczne, poprawne i skończone. W późniejszych czasach jednak, o ile przez 2000 lat geometria była nadal rozwijana, o tyle algorytmika została na długo zapomniana. Częściowym wyjaśnieniem takiego stanu rzeczy może być skuteczność innej metody dowodzenia — dowodzenia przez *sprowadzenie do absurdu*. Metoda ta ułatwia dowodzenie, że jakiś obiekt istnieje. Dowód prowadzi się przez zaprzeczenie, nie podaje się natomiast konkretnego sposobu konstrukcji (czyli algorytmu).

Konstrukcje euklidesowe jeszcze z innego powodu zasługują na uwagę: opisano w nich zestaw środków, z jakich wolno korzystać (linijka i cyrkiel), oraz zdefiniowano zestaw dopuszczalnych czynności elementarnych. Starożytni najbardziej zainteresowani byli tym, czy przyjęte czynności elementarne są domknięte ze względu na konstrukcje skończone. Szczególnie interesowało ich, czy możliwe jest za ich pomocą zrealizowanie wszystkich dających się wymyślić konstrukcji geometrycznych, takich jak trysekcja kąta. Dzisiaj to samo pytanie moglibyśmy sformułować inaczej: czy elementarne czynności konstrukcji euklidesowych wystarczą do wykonania wszystkich możliwych „obliczeń" geometrycznych? Próbując odpowiedzieć na to pytanie, rozważano różne alternatywne metody, z wykorzystaniem innych operacji i innych narzędzi. Archimedes przedstawił (poprawną zresztą) konstrukcję trysekcji kąta 60°, przy czym skorzystał z następującego rozszerzenia zbioru

dopuszczalnych operacji: *Jeśli mamy dwa okręgi, A i B, oraz punkt P, wolno nam na linialе zaznaczyć odcinek MN i umieścić go tak, aby linial przechodził przez P, stykając się z okręgiem A w punkcie M i z okręgiem B w punkcie N*. Czasami badano też ograniczony zbiór narzędzi, na przykład rozważano posługiwanie się jedynie cyrklami. Tego typu próby kojarzyć się muszą z teorią automatów, w której siłę poszczególnych modeli obliczeniowych badamy, nakładając różne ograniczenia. Niestety, dowód niezupełności zbioru narzędzi euklidesowych pojawił się dopiero po stworzeniu i rozwinięciu algebry.

Wpływ „Elementów" Euklidesa na przyszłe pokolenia był tak duży, że aż do czasów Kartezjusza nikt nie proponował nawet innego sformułowania geometrii. Dopiero Kartezjusz, wprowadzając swój układ współrzędnych, umożliwił skorzystanie z osiągnięć algebry, co dopiero pozwoliło zająć się krzywymi wyższych rzędów i otworzyło drogę dla rachunku Newtona. Użycie współrzędnych znacznie zwiększa możliwości obliczeniowe, zasypując przepaść między algebrą i geometrią. Metody obliczeniowe oznaczały też renesans myślenia konstruktywistycznego. Możliwe stało się tworzenie nowych figur geometrycznych przez rozwiązywanie powiązanych z nimi równań. Już wkrótce znów pojawiły się pytania o obliczalność. Gauss, korzystając z narzędzi algebraicznych, wykazał, które wielokąty foremne mające liczbę boków wyrażonych liczbą pierwszą można zbudować, korzystając z narzędzi euklidesowych. Jasne stało się, że konstrukcje wykonywane za pomocą linijki i cyrkla, wyliczanie pól i równania algebraiczne są ściśle ze sobą powiązane. W swej rozprawie doktorskiej Gauss wykazał, że każde równanie algebraiczne ma przynajmniej jeden pierwiastek (jest to podstawowe twierdzenie algebry). W 1828 roku Abel rozważał ten sam problem w ograniczonym modelu obliczeniowym: badał, czy pierwiastek każdego równania algebraicznego można wyznaczyć, korzystając jedynie z operacji arytmetycznych i pierwiastkowania n-tego stopnia. Udowodnił, że jest to niemożliwe. O ile wszystkie liczby dające się skonstruować geometrycznie są algebraiczne[1], o tyle Abel wykazał, że nie wszystkie liczby algebraiczne dają się skonstruować. Niedługo później Abel pokazał, które równania algebraiczne można rozwiązać za pomocą pierwiastków, a dzięki temu możliwe było badanie wykonalności różnych problemów geometrycznych, takich jak trysekcja kąta.

1.1.1. Złożoność geometrii klasycznej

Wszystkie konstrukcje euklidesowe — poza najprostszymi — są bardzo złożone, a to z powodu elementarności dozwolonych operacji. W przeszłości wielokrotnie próbowano udoskonalić geometrię poprzez tworzenie konstrukcji składających się z mniejszej liczby operacji elementarnych. Jednak aż do dwudziestego wieku nie potrafiono określić miary złożoności konstrukcji. W 1902 roku Emile Lemoine w dziele *Géométrographie* następująco scharakteryzował operacje elementarne geometrii euklidesowej [Lemoine (1902)]:

[1] Kiedy mówimy, że „liczba daje się skonstruować", mamy na myśli możliwość konstrukcyjnego zbudowania odcinka o długości wyrażonej taką liczbą, kiedy dany jest odcinek jednostkowy — *przyp. tłum.*

(1) Umieszczenie nóżki cyrkla w danym punkcie.
(2) Umieszczenie nóżki cyrkla na danej prostej.
(3) Narysowanie okręgu.
(4) Przyłożenie brzegu linijki do danego punktu.
(5) Narysowanie prostej.

Łączną liczbę takich operacji wykonywanych podczas tworzenia konstrukcji nazywamy *złożonością*. Definicja taka jest bardzo bliska pojęciu złożoności obliczeniowej algorytmów, choć Lemoine nie powiązał bezpośrednio rozmiaru danych wejściowych (liczby danych punktów i prostych) ze złożonością konstrukcji geometrycznej. Lemoine chciał poprawić starsze konstrukcje euklidesowe, a nie tworzyć teorię ich złożoności. Wcześniej odniósł niejeden sukces; euklidesowe rozwiązanie problemu okręgów Apoloniusza wymaga 508 kroków, zaś rozwiązanie podane przez Lemoine'a liczyło poniżej dwustu kroków [Coolidge (1916)]. Niestety, Lemoine nie dostrzegł wagi dowodu, że dana konstrukcja wymaga pewnej minimalnej liczby kroków.

Znaczenie takiego dolnego ograniczenia liczby kroków dostrzegł Hilbert. Pracując w ograniczonym modelu, rozważał jedynie te konstrukcje, które da się zrealizować za pomocą liniału i *miarki* — narzędzia służącego jedynie do odkładania na prostej odcinka o zadanej długości. Nie do wszystkich konstrukcji euklidesowych taki zestaw narzędzi wystarcza. Jeśli konstrukcja daje się tak przeprowadzić, na współrzędne punktów można patrzeć jako na funkcję F danych punktów. Hilbert podał warunki konieczny i wystarczający do tego, aby F dawała się wyliczyć za pomocą n operacji pierwiastkowania drugiego stopnia; jest to jedno z pierwszych twierdzeń dotyczących algebraicznej złożoności obliczeniowej [Hilbert (1899)].

Wiele wskazuje na to, że różne techniki stosowane dzisiaj do analizy algorytmów były używane przez geometrów już od wieków. W roku 1672 Georg Mohr wykazał, że każda konstrukcja wykonalna linijką i cyrklem może być też zrobiona samym cyrklem, o ile zgodzimy się, aby wszystkie dane i konstruowane obiekty były wyznaczane punktami. Samym cyrklem nie można narysować linii prostej, ale można ją wskazać za pomocą dwóch punktów, powstających przy przecinaniu się okręgów. W dowodzie Mohra ważne jest to, że jest to *symulacja*, pozwalająca pokazać, że każdą operację, w której używamy linijki, można zastąpić skończoną liczbą operacji wykonywanych samym cyrklem. Można by zapytać, czy istnieje tu jakiś ściślejszy związek z teorią automatów. Podobnego typu wnioskiem jest stwierdzenie, że w konstrukcjach, w których używana jest linijka, można użyć linijki dowolnie małej długości, byle większej od zera.

Lemoine i jego naśladowcy zajmowali się złożonością konstrukcji euklidesowych, ale warto też zapytać o to, jakiej *przestrzeni* te konstrukcje wymagają. Naturalną miarą potrzebnej przestrzeni jest jej powierzchnia. Używana przestrzeń zależy od powierzchni wielokąta wypukłego obejmującego potrzebne punkty, powierzchni spodziewanego wyniku oraz powierzchni potrzebnej podczas konstrukcji obiektów pomocniczych [Eves (1972)]. Co warte odnotowania, zapis czasu i powierzchni nie są geometrii całkiem obce.

Wprawdzie Galois wykazał niewykonalność pewnych konstrukcji euklidesowych, wobec czego niemożliwa była na przykład *dokładna* trysekcja kąta, nie wpływa to jednak na możliwość realizacji konstrukcji przybliżonych. Tak naprawdę zbieżne asymptotycznie procedury kwadratury koła i podwojenia sześcianu znane były już starożytnym Grekom [Heath (1921)]. Jak widać, algorytmy iteracyjne mają już naprawdę długą historię.

1.1.2. Teoria zbiorów wypukłych, geometria metryczna i kombinatoryczna

W dziewiętnastym wieku geometria rozwijała się w wielu różnych kierunkach. Jeden z tych kierunków, zapoczątkowany przez Kleina, dotyczył badań nad zachowaniem się obiektów geometrycznych poddanych różnym przekształceniom. Innym ważnym kierunkiem rozwoju była geometria rzutowa (zobacz punkt 1.3.2). Badanie skończonych przestrzeni rzutowych prowadzi do fascynujących pytań z dziedziny kombinatoryki i algorytmów dyskretnych, ale nie będziemy w tej książce zajmować się tymi dziedzinami geometrii.

Rozwój analizy rzeczywistej miał duży wpływ na geometrię, gdyż dał formalne podstawy pojęć, które wcześniej były jedynie intuicyjne. Dwa stworzone tak dzieła — geometria metryczna i teoria wypukłości — stanowią bardzo ważne narzędzia matematyczne pozwalające projektować szybkie algorytmy.

Odległość to jedno z podstawowych pojęć geometrii. Jego uogólnieniem jest *metryka*, która pozwala wprowadzić zagadnienia i metody geometryczne do analizy; „odległość" między funkcjami pozwala tworzyć przestrzenie funkcji i inne zaawansowane konstrukcje. Niestety, wiele twierdzeń tej dziedziny to twierdzenia niekonstrukcyjne. Przestrzenie funkcyjne z samej swej natury nie poddają się obliczeniom.

Znaczenie teorii wypukłości polega na tym, że zajmujemy się właściwościami *globalnymi*, co pozwala rozwiązywać problemy ekstremów. Niestety, wiele zagadnień trudno jest sformułować algebraicznie i znów wiele twierdzeń to twierdzenia niekonstrukcyjne.

Geometria kombinatoryczna w swej naturze jest znacznie bliższa geometrii algorytmicznej. Obiekty geometryczne są w niej charakteryzowane przez właściwości podzbiorów *skończonych*. Przykładowo, zbiór jest wypukły wtedy i tylko wtedy, gdy odcinek wyznaczony przez dowolne dwa punkty tego zbioru jest w tym zbiorze całkowicie zawarty. Nieprzydatność geometrii kombinatorycznej do naszych rozważań wynika stąd, że zwykle liczba skończonych podzbiorów jest nieskończona, co wyklucza podejście algebraiczne. Ostatnie prace nad algorytmami geometrycznymi zmierzały do usunięcia tych niedogodności i stworzenia matematyki dającej w wyniku dobre algorytmy.

1.1.3. Wcześniejsze prace

Algorytmy geometryczne były tworzone w różnych kontekstach, zaś terminu „geometria obliczeniowa" używano przynajmniej w dwóch różnych znaczeniach. Teraz postaramy się wszystkie te wysiłki ułożyć we właściwy sposób i pokazać ich miejsce w dzisiejszej nauce.

(1) Modelowanie geometryczne za pomocą krzywych i powierzchni składanych. Zagadnienie to bliższe jest analizie numerycznej niż geometrii. Największe sukcesy odnieśli tutaj Bézier, Forrest i Riesenfeld. Warto zauważyć, że Forrest tę właśnie dziedzinę wiedzy określa mianem „geometrii obliczeniowej" [Bézier (1972), Forrest (1971), Riesenfeld (1973)].

(2) We wspaniałej książce *Perceptrons* Minsky i Papert (1969) opisali złożoność predykatów pozwalających rozpoznawać pewne własności geometryczne, takie jak wypukłość.

Celem ich pracy było ustalenie, czy możliwe jest użycie dużych czujników, składających się z prostych układów, do rozpoznawania wzorców. Ich teoria jest samodzielna i nie mieści się w tematyce tej książki.

(3) Oprogramowanie graficzne i edytory rysunków są niewątpliwie miejscem, w którym stosowanych jest szereg algorytmów przedstawianych w tej książce. Jednak w tym wypadku pojawia się wiele zagadnień związanych bezpośrednio z implementacją i interfejsem użytkownika, a nie analizą algorytmów. Do tej samej klasy rozwiązań zaliczyć należy oprogramowanie sterujące obrabiarkami numerycznymi, oprogramowanie ploterów, systemy kartograficzne oraz oprogramowanie inżynierskie i architektoniczne.

(4) Pojęcie „geometria obliczeniowa" wielu osobom może kojarzyć się z dowodzeniem twierdzeń geometrycznych za pomocą komputera. Jest to fascynująca tematyka, ale znacznie więcej mówi ona o metodach dowodzenia twierdzeń niż o samej geometrii, więc nie będziemy się nią dalej zajmować.

1.1.4. Ku geometrii obliczeniowej

Na to, co dzisiaj rozumiemy przez geometrię obliczeniową, złożyło się wiele obszarów zastosowań, w których konieczne było opracowanie wydajnych algorytmów rozwiązywania problemów geometrycznych. Przykładowe zagadnienia to problem komiwojażera, minimalne drzewo, ukrywanie linii czy programowanie liniowe, a także mnóstwo innych. Aby pokazać przekonująco różnorodne przykłady zastosowania geometrii obliczeniowej, przed zaprezentowaniem tych problemów najpierw przedstawimy potrzebne informacje pomocnicze.

W literaturze naukowej algorytmiczne studia nad wymienionymi i podobnymi problemami pojawiały się w ciągu całego wieku, a szczególnie często w ostatnim dwudziestoleciu. Jednak dopiero ostatnio podjęto systematyczne badania algorytmów geometrycznych, a zarazem zwiększyło się zainteresowanie tą dziedziną, w publikacji M. I. Shamosa (1975a) nazwaną „geometrią obliczeniową".

Mamy nadzieję, że z omawianych w tej książce zagadnień wyłoni się całościowy obraz geometrii obliczeniowej i stosowanych w niej metod. Jedną z podstawowych cech tej dziedziny jest stwierdzenie, że tradycyjne pojmowanie obiektów geometrycznych niejednokrotnie nie nadaje się do projektowania optymalnych algorytmów. Aby tego problemu uniknąć, konieczne jest odnalezienie pojęć, które są przydatne w obliczeniach, i określenie ich właściwości. Krótko mówiąc, geometria obliczeniowa musi przekształcać w miarę potrzeb klasyczną dziedzinę wiedzy w jej postać obliczeniową.

1.2. Wprowadzenie do algorytmów

W ciągu ostatnich piętnastu lat analiza i projektowanie algorytmów programów było jedną z najdynamiczniej rozwijających się dziedzin informatyki. Fundamentalne prace Knutha (1968, 1973) oraz Aho, Hopcrofta i Ullmana (1974) wprowadziły porządek w bogatym

zbiorze odrębnych wyników, określiły podstawowe paradygmaty i ustaliły metodologię, która stała się standardem. Dalsze prace [Reingold-Nievergelt-Deo (1977), Wirth (1976)] jeszcze bardziej wzmocniły podstawy teoretyczne.

Dokładne omawianie tych doskonałych prac wykracza poza zakres niniejszej książki, ale zakładamy, że tematyka ta jest już znana Czytelnikowi. Jednak warto, choćby z uwagę na terminologię, krótko omówić składniki języka, którym będziemy opisywali geometrię obliczeniową. Składniki te to algorytmy i struktury danych. Algorytmy to programy przeznaczone do wykonywania na odpowiedniej abstrakcji maszyn von Neumanna. Struktury danych to sposoby ułożenia informacji, które w połączeniu z algorytmami pozwalają wydajnie i elegancko rozwiązywać problemy obliczeniowe.

1.2.1. Algorytmy: zapis i określanie wydajności

Algorytmy zapisuje się z uwzględnieniem konkretnego modelu obliczeniowego. Jak zobaczymy w podrozdziale 1.4, model obliczeniowy to wygodna abstrakcja maszyny fizycznej stosowanej do wykonywania programów. Jak jednak wykazali Aho-Hopcroft-Ullman (1974), nie jest konieczne ani pożądane zapisywanie algorytmu w formie kodu maszynowego. Zamiast tego, aby zachować jasność, zwięzłość i zapewnić zrozumiałość zapisu, należy zwykle[2] użyć wysokopoziomowego języka *Pidgin Algol*, który stał się już standardem w literaturze przedmiotu. Pidgin Algol to nieformalna i elastyczna wersja języka podobnego do Algolu. Zapis struktur kontrolnych jest ściśle określony, ale wszelkie inne wyrażenia można zapisywać bardzo dowolnie, ścisły zapis matematyczny może być mieszany z językiem naturalnym. Oczywiście, programy w języku Pidgin Algol mogą być automatycznie przekładane na programy zapisane w formalnych językach wysokiego poziomu.

Podobnie jak Aho-Hopcroft-Ullman, pokażemy konstrukcje języka Pidgin Algol. Formalne deklaracje typów danych nie mają tu zastosowania, zaś typ zmiennej zwykle jasno wynika z kontekstu. Poza tym nie ma żadnego specjalnego formatu zapisu wyrażeń i warunków.

Program nazywamy **procedurą**, ma on postać:

```
procedure nazwa (parametry) instrukcja.
```

Instrukcję można zapisać jako łańcuch dwóch lub więcej instrukcji, ujęty w „nawiasy" „begin...end":

```
begin instrukcja;
        ...
        instrukcja
end.
```

[2] Czasami algorytmy mogą być zapisywane w formie zwykłego tekstu.

Instrukcja może też mieć postać zdania języka naturalnego lub jedną z poniższych postaci:

(1) Przypisanie:

```
zmienna := źródło wartości
```

„Źródło wartości" to opis wyliczenia wartości, która ma być przypisana zmiennej. Wyliczenie to, ogólnie rzecz biorąc, *wyrażenie* na zbiorze zmiennych (niektóre z tych zmiennych mogą być zapisane jako „funkcje"; funkcja to przypadek szczególny programu, co pokażemy dalej).

(2) Instrukcja warunkowa:

```
if warunek then instrukcja (else instrukcja)
```

Część z else jest opcjonalna.

(3) Pętla — może mieć jedną z trzech postaci:

```
3a. for zmienna := wartość until wartość do instrukcja
3b. while warunek do instrukcja
3c. repeat instrukcja until warunek
```

Konstrukcje while i repeat różnią się o tyle, że w przypadku pętli while warunek jest sprawdzany *przed* wykonaniem instrukcji, zaś w pętli repeat najpierw wykonywana jest instrukcja.

(4) Instrukcja powrotu:

```
return wyrażenie
```

Instrukcja tego typu musi pojawić się w programach będących funkcjami. Programy takie mają postać:

```
function nazwa (parametry) instrukcja.
```

Wyrażenie będące argumentem return staje się źródłem wartości dla instrukcji przypisania, pokazanej powyżej w punkcie 1.

Algorytmy zapisane w języku Pidgin Algol często zawierają komentarze, mające ułatwiać ich zrozumienie. Zwykle komentarz zapisuje się w postaci: (*zdanie w języku naturalnym*).

Czas potrzebny na obliczenia — czyli wykonanie algorytmu — to suma czasów wykonywania poszczególnych operacji (zobacz też podrozdział 1.4). Jak już wcześniej wspomnieliśmy, w Pidgin Algolu program można łatwo przekształcić na kod maszynowy

wybranego komputera (choć zadanie to może być żmudne). W zasadzie byłaby to metoda wyznaczenia czasu wykonywania programu. Jednak jest to rozwiązanie nie tylko żmudne, ale i mało kształcące, gdyż odwołujemy się do konkretnego komputera, podczas gdy interesuje nas zależność funkcyjna między czasem wykonania a wielkością rozwiązywanego problemu (czyli jak szybko rośnie czas wykonania przy wzroście wielkości problemu). Wobec tego przyjęło się w analizie i projektowaniu algorytmów określać czas wykonywania (a także wszelkie inne miary szybkości działania) z pominięciem stałych współczynników. Zwykle robi się to, uwzględniając jedynie wybrane „operacje kluczowe" algorytmu (wystarcza do tego analiza algorytmu zapisanego w języku wysokiego poziomu). Takie podejście jest jak najbardziej uprawnione, jeśli chodzi o ustalanie dolnego ograniczenia czasu wykonania, gdyż wszystkie nieuwzględniane w takim przypadku operacje mogą czas wykonania jedynie zwiększać. Jeśli jednak chodzi o górne ograniczenie, trzeba zapewnić, że wybrane operacje będą stanowiły stały ułamek wszystkich operacji algorytmu. Knuth spopularyzował metodę zapisu, która pozwala rozróżnić ograniczenie górne i dolne; w tej książce także skorzystamy z tej metody [Knuth (1976)]:

- $O(f(N))$ oznacza zbiór wszystkich funkcji $g(N)$ takich, że istnieją stałe dodatnie C i N_0, że dla wszystkich $N \geq N_0$ zachodzi $|g(N)| \leq Cf(N)$.
- $\Omega(f(N))$ oznacza zbiór wszystkich funkcji $g(N)$ takich, że istnieją dodatnie stałe C i N_0 takie, że dla wszystkich $N \geq N_0$ zachodzi $g(N) \geq Cf(N)$.
- $\theta(f(N))$ oznacza zbiór wszystkich funkcji $g(N)$ takich, że istnieją dodatnie stałe C_1, C_2 i N_0 takie, że dla wszystkich $N \geq N_0$ zachodzi $C_1f(N) \leq g(N) \geq C_2f(N)$.
- $o(f(N))$ oznacza zbiór wszystkich funkcji $g(N)$ takich, że *dla wszystkich dodatnich stałych* C istnieje stała N_0 taka, że dla wszystkich $N \geq N_0$ zachodzi $g(N) \leq Cf(N)$ (lub, co jest równoważne, $\lim_{N\to\infty} g(N)/f(N) = 0$).

Tak więc $O(f(N))$ zawiera funkcje *nie większe* niż pewne stałe razy $f(N)$ — tak można określić ograniczenie górne. Z kolei $\Omega(f(N))$ zawiera funkcje *nie mniejsze* niż pewna stała razy $f(N)$, więc mamy ograniczenie dolne. W końcu $\theta(f(N))$ zawiera funkcje tego samego rzędu co $f(N)$, przez co możemy określić algorytmy „optymalne".

Omówiliśmy już czas wykonywania algorytmu. Inną ważną miarą wydajności jest zajmowanie pamięci. Właśnie *pamięć* i *czas*, wyrażone jako funkcje wielkości problemu, stanowią dwie podstawowe miary wydajności przy analizie algorytmów.

Głównym celem niniejszej książki jest przedstawienie algorytmów dotyczących problemów geometrycznych i oszacowanie ich złożoności *w najgorszym przypadku*. Złożoność w najgorszym przypadku to największa miara wydajności algorytmu przy danej wielkości problemu. Z kolei złożoność *przypadku średniego* (czyli złożoność *oczekiwana*) to oszacowanie, jakiej złożoności powinniśmy spodziewać się podczas testów. Niestety, analiza przypadku średniego jest znacznie trudniejsza od analizy przypadku najgorszego. Po pierwsze, pojawiają się trudności matematyczne — nawet w przypadku dobrego doboru rozkładu parametrów. Po drugie, rzadko udaje się ustalić, jak w praktyce będzie wyglądało typowe użycie algorytmu. Dlatego właśnie zwykle analizuje się przypadek najgorszy. W niniejszej książce sporadycznie zajmować się będziemy wynikami przypadku średniego.

Inną ważną rzeczą, którą należy tutaj podkreślić, jest fakt, że w zapisie „rzędu złożoności" nie pojawiają się współczynniki mnożenia. Wobec tego uzyskana złożoność obowiązuje jedynie w przypadku dostatecznie dużych problemów. Z tego powodu mówimy

w takiej sytuacji o *analizie asymptotycznej*. Możliwe jest — i wcale nie rzadkie — że w przypadku małych problemów optymalny algorytm nie jest algorytmem najlepszym asymptotycznie. Wybierając algorytm do konkretnego zastosowania, trzeba powyższe zawsze mieć na uwadze.

1.2.2. Nieco o technikach tworzenia algorytmów

Wydajne algorytmy opisujące problemy geometryczne często tworzy się, przetwarzając techniki ogólne danej dziedziny, jak „dziel i rządź", równoważenie, rekurencję czy programowanie dynamiczne. Doskonałe omówienie tych technik znaleźć można w klasycznych już tekstach poświęconych analizie i projektowaniu algorytmów (na przykład [Aho-Hopcroft-Ullman (1974)]) i zbędne jest tutaj powtarzanie tego.

Istnieje technika, która w sposób naturalny jest szczególnie polecana do rozwiązywania problemów geometrycznych. Technikę tę nazywamy *wymiataniem*, przy czym najczęściej korzysta się z *wymiatania płaszczyzny* (w dwóch wymiarach) i *wymiatania przestrzeni* (w trzech wymiarach). Teraz omówimy najważniejsze cechy wymiatania płaszczyzny; uogólnienie tego do trzech wymiarów jest już proste.

Aby nie być gołosłownym, pokażemy tę metodę na konkretnym przykładzie (który dokładniej omówimy w punkcie 7.2.3): jeśli mamy dany zbiór odcinków na płaszczyźnie, należy znaleźć wszystkie ich przecięcia. Rozważmy prostą l (bez utraty ogólności możemy założyć, że jest ona pionowa), która dzieli płaszczyznę na lewą i prawą półpłaszczyznę. Załóżmy, że każda z tych półpłaszczyzn zawiera końce danych odcinków. Jest jasne, że rozwiązaniem naszego zadania będzie suma rozwiązań dla wszystkich takich par półpłaszczyzn. Zakładając zatem, że mamy już zbiór przecięć na lewo od l, wiemy, że na uzyskany zbiór nie będą miały wpływu odcinki znajdujące się na prawo od l. Zauważmy, że przecięcie może wystąpić jedynie między dwoma takimi odcinkami, których przecięcia z pewną pionową prostą przylegają do siebie. Tak więc, jeśli uwzględnimy *wszystkie* pionowe przecięcia danego zbioru odcinków, odnajdziemy wszystkie przecięcia. Jako że jednak niemożliwe jest wyliczenie ciągłego, nieskończonego zbioru wszystkich przecięć pionowych, musimy ten problem rozwiązać inaczej. Zauważmy, że płaszczyzna dzielona jest na pionowe paski, z których każdy ograniczony jest albo końcami odcinków, albo przecięciami odcinków, przy czym pionowa kolejność odcinania przez pionowe cięcie jest stała. Wobec tego wystarczy przejść od lewego brzegu takiego paska do jego brzegu prawego, zaktualizować kolejność odcinków i sprawdzić, czy między „przylegającymi" odcinkami występują jakieś nowe przecięcia.

W powyższym omówieniu pokazaliśmy podstawowe cechy techniki wymiatania płaszczyzny. Pionowa prosta przesuwana jest po płaszczyźnie od lewej do prawej, przy czym zatrzymuje się w punktach szczególnych, nazywanych „punktami zdarzeń". Przecięcie tej prostej z badanymi danymi zawiera wszystkie informacje *potrzebne* do przejścia dalej. Tak więc mamy dwie podstawowe struktury:

(1) *Harmonogram punktów zdarzeń*, który określa ciąg współrzędnych odciętych, uporządkowany od lewej do prawej — w ten sposób definiujemy punkty zatrzymania linii wymiatającej. Zwróćmy uwagę, że harmonogram punktów zdarzeń nie musi

od razu wynikać z danych wejściowych, ale może być aktualizowany dynamicznie podczas wykonywania algorytmu wymiatania płaszczyzny. W różnych zastosowaniach potrzebne mogą być różne struktury danych.

(2) *Stan prostej wymiatającej*, który odpowiada opisowi przecięcia linii wymiatającej z geometryczną strukturą wymiataną. Owo odpowiadanie oznacza, że przecięcie zawiera informacje charakterystyczne dla danego zastosowania. Stan prostej wymiatającej jest aktualizowany przy przyjściu do każdego następnego punktu; każdorazowo konieczne jest też wybieranie odpowiedniej struktury danych.

Przykłady algorytmów wymiatania płaszczyzny pokażemy w punkcie 2.2.2.

1.2.3. Struktury danych

W algorytmach geometrycznych mamy do czynienia z przetwarzaniem obiektów, które nie są obsługiwane bezpośrednio na poziomie języka maszynowego. Wobec tego użytkownik musi te złożone obiekty opisać za pomocą prostszych typów danych, które są dla komputera zrozumiałe. Opisy tego typu określamy mianem *struktur danych*.

Najpowszechniej spotykanymi obiektami złożonymi w algorytmach geometrycznych są zbiór i ciąg (uporządkowany zbiór). Struktury danych najlepiej je opisujące omówiono w literaturze podstawowej o algorytmach i do niej kierujemy Czytelnika [Aho-Hopcroft-Ullman (1974), Reingold-Nievergelt-Deo (1977)]. Nam wystarczy jedynie wymienienie dostępnych struktur danych oraz opisanie ich działania i wpływu na wydajność.

Niech S będzie zbiorem reprezentowanym przez strukturę danych, a u niech będzie dowolnym elementem zbioru, którego S jest podzbiorem. Podstawowe operacje na zbiorach to:

(1) MEMBER(u, S). Czy $u \in S$? (odpowiedź: TAK lub NIE).
(2) INSERT(u, S). Dodanie u do S.
(3) DELETE(u, S). Usunięcie u z S.

Załóżmy, że $\{S_1, S_2, \ldots, S_k\}$ to zbiór zbiorów (parami rozłącznych). Przydatnymi operacjami na takim zbiorze zbiorów są:

(4) FIND(u). Podanie j, jeśli $u \in S_j$.
(5) UNION(S_i, S_j; S_k). Tworzona jest suma zbiorów S_i i S_j, oznaczana jako S_k.

Kiedy zbiór zewnętrzny jest całkowicie uporządkowany, bardzo ważne są następujące operacje:

(6) MIN(S). Podanie najmniejszego elementu z S.
(7) SPLIT(u, S). Zbiór S dzielony jest na $\{S_1, S_2\}$ takie, że $S_1 = \{v: v \in S \text{ oraz } v \leq u\}$ oraz $S_2 = S - S_1$.
(8) CONCATENATE(S_1, S_2). Zakładając, że dla dowolnych $u' \in S_1$ i $u'' \in S_2$ mamy $u' \leq u''$, tworzymy uporządkowany zbiór $S = S_1 \cup S_2$.

Struktury danych można pogrupować ze względu na operacje, jakie są w nich możliwe (abstrahując od szybkości wykonywania tych operacji). Tak więc w przypadku zbiorów uporządkowanych mamy następującą tabelę:

TABELA I

Struktura danych	Obsługiwane operacje
Słownik	MEMBER, INSERT, DELETE
Kolejka priorytetowa	MIN, INSERT, DELETE
Kolejka łączona	INSERT, DELETE, SPLIT, CONCATENATE

Z uwagi na wydajność wszystkie te struktury danych zwykle realizuje się jako zbilansowane drzewo binarne (często AVL albo 2- lub 3-drzewo) [Aho-Hopcroft-Ullman (1974)]. W ten sposób każda operacja jest wykonywana w czasie proporcjonalnym do logarytmu liczby elementów zapisanych w strukturze danych. Ilość zajmowanej pamięci jest proporcjonalna do wielkości zbioru.

Na powyższe struktury danych można spojrzeć bardziej abstrakcyjnie, jako na liniowe tablice (czyli *listy*), gdzie wstawienia i usunięcia można wykonywać na dowolnych pozycjach takiej tablicy. W niektórych zastosowaniach bardziej odpowiednie są ograniczone, uproszczone tryby dostępu. Strukturami takimi są: *kolejki*, w których wstawianie ma miejsce na końcu, a usunięcie na drugim końcu; *stosy*, gdzie wstawianie i usuwanie ma miejsce na jednym końcu (szczycie stosu). Jasne jest, że zarządzanie stosem i kolejką wymaga użycia odpowiednio jednego i dwóch wskaźników. Aby skrócić zapis, mając na myśli dodanie i usunięcie z U, będziemy używać odpowiednio symboli „$\Rightarrow U$" oraz „$U \Rightarrow$", gdzie U jest kolejką lub stosem.

Zbiory nieuporządkowane zawsze można traktować tak samo, jak zbiory uporządkowane, sztucznie ustalając porządek elementów (na przykład przypisując elementom „nazwy" i stosując kolejność alfabetyczną). Typową strukturą danych w takim przypadku jest:

TABELA II

Struktura danych	Obsługiwane operacje
Sterta łączona	INSERT, DELETE, FIND, UNION, (MIN)

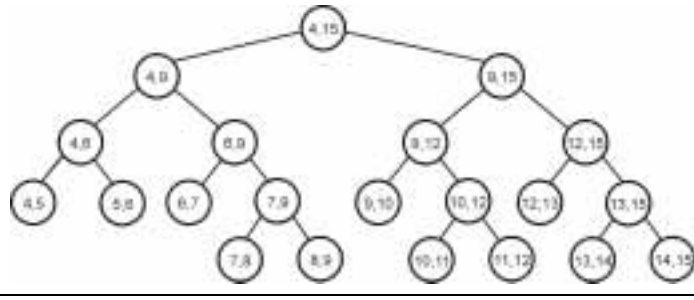
Każda z powyższych operacji wykonywana jest w czasie $O(\log N)$, gdzie N jest rozmiarem zbioru w strukturze danych, przy czym całość implementowana jest jako zrównoważone drzewo. Jeśli elementy rozważanego zbioru będą reprezentowane jako liczby całkowite od 1 do N, możliwe jest dopracowanie struktury danych tak, aby N operacji na zbiorze o rozmiarze N było wykonywanych w czasie $O(N \cdot A(N))$, gdzie $A(N)$ jest wyjątkowo wolno rosnącą funkcją, związaną z funkcją Ackermanna (przykładowo dla $N \leq 2^{2^{16}}$, czyli $\sim 10^{20\,000}$, $A(N) \leq 5$).

Przedstawione powyżej typowe struktury danych są bardzo często używane w algorytmach geometrii obliczeniowej. Jednak z natury problemów geometrycznych wynikła konieczność stworzenia specyficznych, nietypowych struktur danych, z których dwie są tak powszechnie stosowane i przydatne, że na miejscu będzie ich opisanie w tym rozdziale wstępnym. Są to drzewo przedziałów i podwójnie powiązana lista krawędzi.

1.2.3.1. Drzewo przedziałów

Drzewo przedziałów, pierwotnie wprowadzone przez J. L. Bentleya [Bentley (1977)], jest strukturą danych, służącą do zapisu przedziałów prostej rzeczywistej, których końce należą do *ustalonego* zbioru N odciętych. Jako że zbiór odciętych jest ustalony, drzewo przedziałów jest strukturą *statyczną* z uwagi na odcięte, których nie można dodawać ani usuwać. Poza tym odcięte mogą być znormalizowane przez zastąpienie ich numerem liczonym od lewej do prawej. Nie tracąc ogólności, można rozważać te odcięte jako liczby całkowite z zakresu $[1, N]$.

Drzewo odcinków to drzewo binarne z korzeniem. Jeśli mamy liczby całkowite l i r, przy czym $l < r$, drzewo przedziałów $T(l, r)$ buduje się rekurencyjnie: ma ono korzeń v, parametry $B[v] = l$ oraz $E[v] = r$ (B i E to skróty od angielskich słów *Beginning* i *End*, odpowiednio „początek" i „koniec") oraz — jeśli $r–l > 1$ — drzewo to ma lewe poddrzewo $T(l, \lfloor (B[v]+E[v])/2 \rfloor)$ oraz poddrzewo prawe $T(\lfloor (B[v]+E[v])/2 \rfloor, r)$. Korzenie tych dwóch poddrzew oznacza się jako LSON[v] i RSON[v]. Parametry $B[v]$ i $E[v]$ określają *przedział* $[B[v], E[v]] \subseteq [l, r]$, powiązany z węzłem v. Drzewo przedziałów pokazano na rysunku 1.1. Zbiór przedziałów $\{[B[v], E[v]]: v$ jest węzłem $T(l, r)\}$ to *przedziały standardowe* $T(l, r)$. Przedziały standardowe należące do liści $T(l, r)$ nazywamy *przedziałami elementarnymi*[3]. Łatwo wykazać, że $T(l, r)$ jest zrównoważone (wszystkie liście należą do kolejnych przedziałów, a jego wysokość to $\lceil \log_2(r–l) \rceil$.



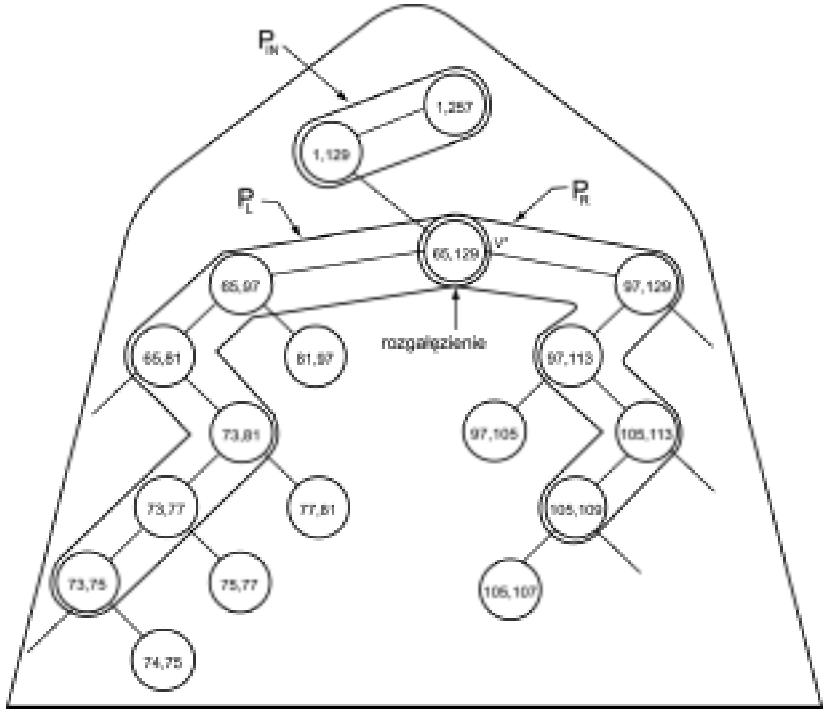
RYSUNEK 1.1. Drzewo przedziałów T(4, 15)

Drzewo przedziałów $T(l, r)$ służy do zapisu przedziałów, których końce należą do zbioru $\{l, l+1, \ldots, r\}$ w sposób *dynamiczny*, czyli z możliwością dodawania i usuwania. W szczególności dla $r–l > 3$ dowolny przedział $[b, e]$ z liczbami całkowitymi $b < e$ zostanie podzielony na zbiór najwyżej $\lceil \log_2(r–l) \rceil + \lfloor \log_2(r–l) \rfloor – 2$ przedziałów standardowych $T(l, r)$. Podział przedziału $[b, e]$ jest całkowicie zdefiniowany przez operację zapisującą (wstawiającą) $[b, e]$ do drzewa T, czyli przez następujące wywołanie INSERT(b, e; korzeń(T)):

```
        procedure INSERT(b, e; v)
begin if (b ≤ B[v]] and (E[v] ≤ e) then wstaw [b, e] do v
      else begin if(b < ⌊(B[v]+E[v])/2⌋) then INSERT(b, e; LSON[v]);
                 if(⌊(B[v]+E[v])/2⌋ < e) then INSERT(b, e;RSON[v]
           end
end.
```

[3] Ściśle rzecz biorąc, przedział związany z v jest częściowo domkniętym przedziałem $[B[v], E[v]]$; wyjątkiem są końcowe prawe węzły $T(l, r)$, których przedziały są domknięte.

Wywołaniu INSERT(b, e; korzeń(T)) odpowiada „przejście" po T, którego ogólna struktura jest taka, jak to pokazano na rysunku 1.2. Ścieżka początkowa może być pusta — oznaczamy ją jako P_{IN}; przechodzi ona od korzenia do węzła v*. Węzeł ten nazywamy *rozgałęzieniem*, gdyż od niego zaczynają się dwie ścieżki P_L i P_R (które mogą być puste). Niezależnie od tego, czy wstawiany przedział jest całkowicie umieszczony w rozgałęzieniu (kiedy P_L i P_R są puste), wszystkie prawe węzły potomne P_L nie należące do P_L, jak również wszystkie lewe węzły potomne P_R nie należące do P_R identyfikują fragmentację [b, e] (*węzły alokacji*).



RYSUNEK 1.2. Wstawianie przedziału [74, 107] do T(1, 257). Węzły alokacji zostały obwiedzione podwójnym kółkiem

Alokacja przedziału w węźle v z T może mieć inną postać, zależną od wymagań w konkretnym zastosowaniu. Często wystarczy znać liczbę elementów zbioru przedziałów alokowanych w danym węźle v; wystarczy do tego pojedynczy parametr $C[v]$, będący nieujemną liczbą całkowitą i oznaczający liczbę elementów; tak więc alokacja [b, e] w v staje się

$$C[v] := C[v]+1.$$

W innych zastosowaniach konieczne jest zachowanie tożsamości przedziałów alokowanych w węźle v. Następnie dołączamy do każdego węzła v z T pomocniczą strukturę — listę $L[v]$, której elementy są identyfikatorami przedziałów.

Operacją symetryczną do INSERT jest DELETE, zapisywana następująco (zakładamy, że interesuje nas zapamiętanie parametru $C[v]$):

```
        procedure DELETE(b, e; v)
begin if (b ≤ B[v]) and (E[v] ≤ e) then C[v] := C[v]–1
      else begin if (b < ⌊(B[v]+E[v])/2⌋) then DELETE(b, e; LSON[v]);
                 if (⌊(B[v]+E[v])/2⌋ < e) then DELETE(b, e; RSON[v]);
           end
end.
```

Zauważmy, że jedynie usuwanie przedziałów wcześniej wstawionych gwarantuje nam poprawność działań.

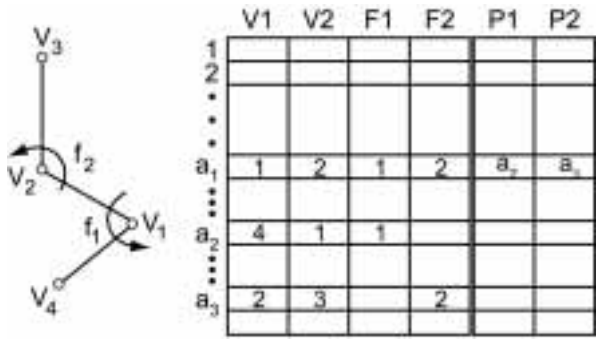
Drzewo przedziałów jest wyjątkowo wszechstronną strukturą danych, o czym przekonać się będziemy mogli w wielu przykładach zastosowań (rozdziały 2. i 8.). Zauważmy jeszcze tylko, że jeśli chcemy wiedzieć, ile przedziałów zawiera dany punkt x, wystarczy zwykłe wyszukiwanie binarne w T (czyli przejście ścieżki od korzenia do liścia).

1.2.3.2. Podwójnie powiązana lista krawędzi

Podwójnie powiązana lista krawędzi (oznaczana jako DCEL, od angielskiej nazwy *doubly-connected-edge-list*) świetnie nadaje się do zapisu grafów płaskich, umieszczonych na płaszczyźnie [Muller-Preparata (1978)]. Zanurzenie w płaszczyźnie grafu płaskiego $G = (V, E)$ to odwzorowanie wszystkich węzłów z V na punkty płaszczyzny i wszystkich krawędzi z E na krzywe łączące obrazy dwóch węzłów, które dana krawędź łączy; odwzorowanie jest tak dobrane, że żadne krawędzie nie przecinają się nigdzie poza swoimi końcami. Wiadomo powszechnie, że wszystkie grafy płaskie można tak zanurzyć w płaszczyźnie, aby wszystkie krawędzie były reprezentowane przez odcinki proste [Fary (1948)].

Niech $V = \{v_1, ..., v_N\}$ oraz $E = \{e_1, ..., e_M\}$. Podstawowym składnikiem listy krawędzi grafu płaskiego (V, E) jest *węzeł krawędzi*. Między krawędziami i węzłami krawędzi istnieje odwzorowanie jeden do jednego, czyli każda krawędź reprezentowana jest dokładnie raz. Węzeł krawędzi zawiera cztery pola z wartościami: *V1*, *V2*, *F1* i *F2* oraz dwa wskaźniki, *P1* i *P2*. Tak więc odpowiednią strukturę danych można bez trudu zaimplementować jako sześć tablic o takich samych nazwach, przy czym każda tablica składa się z M komórek. Znaczenie pól jest następujące: *V1* zawiera początek krawędzi, *V2* jej koniec; tak oto krawędzi standardowo przypisywana jest orientacja. Pola *F1* i *F2* zawierają nazwy opisujące krawędź z *V1* do *V2*: nazwy leżące na lewo i na prawo od krawędzi. Wskaźnik *P1* (odpowiednio *P2*) wskazuje węzeł krawędzi, zawierający pierwszą krawędź znajdującą się za (*V1*, *V2*), przy ruchu w kierunku przeciwnym do ruchu wskazówek zegara. Jako nazwy mogą być użyte liczby całkowite. Przykładowy graf wraz z odpowiadającą mu listą DCEL pokazano na rysunku 1.3.

Łatwo teraz pokazać, jak na liście DCEL można wskazać krawędzie przylegające do danego węzła lub krawędzie zamykające dany obszar. Jeśli graf ma N węzłów i F obszarów, możemy założyć, że mamy do czynienia z dwiema tablicami — *HV*[1:*N*] oraz *HF*[1:*F*] — nagłówków węzłów i listy obszarów: tablice te można wypełnić, przeglądając tablice *V1* i *F1* w czasie $O(N)$. Poniższa procedura VERTEX(j) pozwala uzyskać listę krawędzi przylegających do v_j w formie ciągu adresów z tablicy A.



RYSUNEK 1.3. Przykład listy DCEL

```
        procedure VERTEX(j)
begin a := HV[j];
      a0 := a;
      A[1] := a;
      i := 2;
      if (V1[a] = j) then a := P1[a] else a := P2[a];
      while (a ≠ a0) do
          begin A[i] := a;
                if (V1[a] = j) then a := P1[a] else a := P2[a];
                i := i+1
          end
end.
```

VERTEX(j) wykonuje się w czasie proporcjonalnym do liczby krawędzi przylegających do v_j. Analogicznie możemy stworzyć procedurę FACE(j), pobierającą ciąg krawędzi zamykających f_j; wystarczy zamienić *HV* i *V1* na *HF* i *F1* w powyższej procedurze VERTEX(j). Zauważmy, że procedura VERTEX bada krawędzie w kierunku przeciwnym do ruchu wskazówek zegara, podczas gdy FACE bada je dookoła obszaru w kierunku zgodnym z ruchem wskazówek zegara.

Często graf płaski $G = (V, E)$ zapisuje się jako listę krawędzi, gdzie dla każdego węzła $v_j \in V$ wskazuje się listę krawędzi przylegających ułożonych tak, jak są ułożone dookoła v_j w kierunku przeciwnym do ruchu wskazówek zegara. Łatwo pokazać, że zapis G w formie listy krawędzi można przekształcić na DCEL w czasie $O(|V|)$.

1.3. Podstawy geometrii

1.3.1. Definicje ogólne, przyjęte konwencje

Obiekty, którymi zajmuje się geometria obliczeniowa, to najczęściej zbiory punktów w przestrzeni euklidesowej[4]. Zakłada się użycie prostokątnego systemu współrzędnych, wobec czego każdemu punktowi odpowiada wektor w tych współrzędnych. Obiekty geometryczne nie muszą składać się ze skończonej liczby punktów, ale muszą być *skończenie opisywalne* (zwykle przez skończony łańcuch parametrów). Tak więc poza pojedynczymi punktami, będziemy rozważali proste zawierające dwa dane punkty, odcinki o danych końcach, płaszczyzny zawierające dane trzy punkty, wielokąty określone uporządkowanym ciągiem wierzchołków i tak dalej.

W tym punkcie nie będziemy podawać formalnych definicji opisywanych pojęć; chodzi jedynie o przypomnienie najważniejszych faktów i przedstawienie stosowanego dalej zapisu.

Przez E^d oznaczamy *d-wymiarową przestrzeń euklidesową*, czyli przestrzeń d-krotek $(x_1, \ldots, x_d)$ liczb całkowitych x_i, $i = 1, \ldots, d$ z metryką $(\sum_{i=1}^{d} x_i^2)^{1/2}$. Teraz przypomnimy definicje najważniejszych obiektów, którymi zajmuje się geometria obliczeniowa.

Punkt. d-krotka $(x_1, \ldots, x_d)$ oznacza punkt p w E^d; Punkt ten może być też zinterpretowany jako d-wektor zaczepiony w początku układu współrzędnych E^d, którego swobodnym końcem jest punkt p.

Prosta, płaszczyzna, rozmaitość liniowa. Jeśli dane są dwa różne punkty q_1 i q_2 należące do E^d, to kombinacja liniowa

$$\alpha_1 q_1 + (1 - \alpha) q_2 \qquad (\alpha \in \mathbf{R})$$

jest prostą w E^d. Ogólnie, jeśli danych jest k niezależnych liniowo punktów $q_1, \ldots, q_k$ należących do E^d $(k \le d)$, to kombinacja liniowa

$$\alpha_1 q_1 + \alpha_2 q_2 + \cdots + \alpha_{k-1} q_{k-1} + (1 - \alpha_1 - \cdots - \alpha_{k-1}) q_k$$
$$(\alpha_j \in \mathbf{R}, \quad j = 1, \ldots, k-1)$$

jest *kombinacją liniową* wymiaru $(k–1)$ w E^d.

Odcinek. Jeśli dane są dwa różne punkty q_1 i q_2 w E^d, jeśli do wyrażenia $\alpha q_1 + (1–\alpha) q_2$ dodamy warunek $0 \le \alpha \le 1$, otrzymamy *wypukłą kombinację* q_1 i q_2, czyli:

$$\alpha q_1 + (1 - \alpha) q_2 \qquad (\alpha \in \mathbf{R}, 0 \le \alpha \le 1)$$

[4] Ograniczenie się do przestrzeni euklidesowej (pewnego szczególnego, choć wyjątkowo ważnego przypadku geometrii metrycznej) pozwala nam wykorzystać intuicję, a poza tym to właśnie przestrzeni euklidesowej dotyczy większość zastosowania. Jednak ograniczenie to nie jest tak bardzo ważne w wielu aplikacjach, które będziemy omawiać w dalszych rozdziałach. Do tego ważnego zagadnienia wrócimy jeszcze w punkcie 1.3.2.

Kombinacja taka opisuje *odcinek* łączący punkty q_1 i q_2. Zwykle odcinek oznacza się jako $\overline{q_1 q_2}$ (para nieuporządkowana).

Zbiór wypukły. Figura D w E^d jest wypukła, jeśli dla dowolnych dwóch punktów q_1 i q_2 należących do D odcinek $\overline{q_1 q_2}$ jest całkowicie zawarty w D.

Można wykazać, że przecięcie figur wypukłych zawsze jest figurą wypukłą.

Wypukły wielokąt opisany. *Wypukły wielokąt opisany* zbioru punktów S w E^d to najmniejsza figura wypukła w E^d, zawierająca S.

Wielokąt. W E^2 *wielokąt* definiuje się za pomocą skończonego zbioru odcinków takich, że każdy koniec każdego odcinka jest wspólny dla dokładnie dwóch krawędzi i żaden podzbiór krawędzi nie ma takiej własności. Odcinki są krawędziami, zaś ich końce są wierzchołkami wielokąta (warto zauważyć, że krawędzi i wierzchołków jest tyle samo). Wielokąt o n wierzchołkach to n-kąt.

Wielokąt jest *prosty*, jeśli żadna para niekolejnych krawędzi nie ma wspólnych punktów. Wielokąt prosty dzieli płaszczyznę na dwa rozdzielne podzbiory: *wnętrze* (ograniczone) oraz *zewnętrze* (nieograniczone), które są rozdzielone wielokątem (twierdzenie Jordana o krzywej). Pamiętajmy, że w mowie potocznej mówiąc o wielokącie, mamy zwykle na myśli sumę brzegu i wnętrza.

Wielokąt prosty jest *wypukły*, jeśli wypukłe jest jego wnętrze.

Wielokąt prosty P jest *gwiazdokształtny*, jeśli istnieje punkt z nie zewnętrzny do P taki, że dla wszystkich punktów p z P odcinek $\overline{zp}$ jest całkowicie zawarty w P (tak więc wszystkie wielokąty wypukłe są jednocześnie gwiazdokształtne). Położenie punktu z, mającego opisaną właściwość, określa *jądro* P. Wobec tego wielokąt wypukły jest swoim własnym jądrem.

Graf płaski. Graf $G = (V, E)$ zbiór węzłów V, zbiór krawędzi E) jest *płaski*, jeśli można zanurzyć go w płaszczyźnie unikając przecięć (punkt 1.2.3.2) Zanurzenie grafu płaskiego o prostych krawędziach w płaszczyźnie określa podział płaszczyzny nazywany *podziałem płaskim* lub *mapą*. Niech v, e i f oznaczają odpowiednią liczbę węzłów, krawędzi i obszarów (łącznie z jednym obszarem nieograniczonym) dla podziału. Owe trzy wielkości powiązane są ze sobą klasycznym *wzorem Eulera* [Bollobás (1979)]:

$$v - e + f = 2 \tag{1.1}$$

Jeśli mamy dodatkową właściwość polegającą na tym, że wszystkie węzły są stopnia ≥ 3, to łatwo jest udowodnić następujące nierówności:

$$\begin{cases} v \leq \dfrac{2}{3}e, & e \leq 3v - 6 \\ e \leq 3f - 6, & f \leq \dfrac{2}{3}e \\ v \leq 2f - 4, & f \leq 2v - 4 \end{cases} \tag{1.2}$$

co pokazuje, że v, e i f są parami proporcjonalne (zauważmy też, że trzy nierówności z prawej strony są prawdziwe bezwarunkowo).

Triangulacja. Podział płaski jest *triangulacją*, jeśli obszary są trójkątami. *Triangulacja zbioru skończonego S* jest grafem płaskim *S* z maksymalną liczbą krawędzi (inaczej mówiąc, triangulację *S* uzyskujemy, łącząc punkty *S* nieprzecinającymi się odcinkami tak, aby wszystkie wewnętrzne obszary wypukłego wielokąta opisanego na *S* były trójkątami.

Wielościan. W E^3 *wielościan* definiuje się jako skończony zbiór wielokątów płaskich takich, że każda krawędź dowolnego wielokąta jest wspólna dla niego i dokładnie jednego innego wielokąta (wielokąty przyległe) oraz żaden podzbiór wielokątów nie ma tej samej własności. Wierzchołki i krawędzie wielokątów stają się *wierzchołkami* i *krawędziami* wielościanu, zaś wielokąty są jego *ścianami*.

Wielościan jest *prosty*, jeśli żadna para nieprzylegających ścian nie ma wspólnych punktów. Wielościan prosty dzieli przestrzeń na dwie rozłączne części, *wnętrze* (ograniczone) oraz *zewnętrze* (nieograniczone). (W mowie potocznej przez wielościan zwykle rozumiemy ściany wraz z wnętrzem).

Powierzchnia wielościanu (rzędu zero) jest izomorficzna z podziałem płaszczyzny. Wobec tego liczby węzłów v, krawędzi e i ścian f spełniają wzór Eulera (1.1).

Wielościan prosty jest *wypukły*, jeśli wypukłe jest jego wnętrze.

1.3.2. Niezmienniki grup przekształceń liniowych

Można byłoby potraktować geometrię w pełni aksjomatycznie, jako naukę o zbiorach obiektów: punktów, prostych, płaszczyzn i tak dalej, oraz o relacjach między tymi zbiorami. Takie obiekty nie musiałyby mieć żadnych intuicyjnie zrozumiałych właściwości. Prawidłowe korzystanie z aksjomatów powinno pozwolić wywieść wszystkie właściwości geometrii. Takie podejście zaprezentował jako pierwszy Hilbert pod koniec XIX wieku [Hilbert (1899)]; próba ta miała ogromny wpływ na dalszy rozwój geometrii. Z kolei w przypadku podejścia intuicyjnego, bliższego tradycji, wystarczy pamiętać, że można nawet założyć, że zbiory punktów i prostych są skończone.

Z praktycznego punktu widzenia najlepiej jednak patrzeć na geometrię jako na formalną abstrakcję, opisującą obiekty i zjawiska znane z życia codziennego. Wtedy podstawowymi składnikami teorii są modele oparte na pojęciach zrozumiałych intuicyjnie, mające odpowiedniki fizyczne; przykładowo, linia prosta jest abstrakcją odpowiadającą promieniowi światła, zaś wszystkie twierdzenia jej dotyczące mają swoją interpretację eksperymentalną. Podejście to nie stoi w całkowitej opozycji do podejścia poprzedniego; przypisujemy jedynie aksjomatom intuicyjnie zrozumiałe pojęcia geometryczne. Trzeba jednak zauważyć, że nawet przy takich ograniczeniach tworzona teoria nadal jest bardzo szeroka. W pewnym sensie nawet nieeuklidesowa geometria teorii względności jest wnioskiem z intuicyjnych własności geometrycznych.

Naturalne jest, że omawiając algorytmy ograniczymy się jedynie do przestrzeni euklidesowej. W końcu jest ona bardzo przydatnym modelem, wykorzystywanym w ogromnej liczbie zastosowań aplikacji geometrycznych. W szczególności nie jest przypadkiem wykorzystywanie właśnie płaszczyzn i przestrzeni euklidesowych w najważniejszych zastosowaniach metod geometrycznych: w grafice komputerowej, projektowaniu wspomaganym

komputerowo i innych. Załóżymy też, że używana przestrzeń euklidesowa będzie miała ortonormalny układ współrzędnych[5].

Warto jednak zastanowić się nad możliwościami rozszerzenia poszczególnych wniosków z twierdzeń na przestrzenie inne niż euklidesowe. W szczególności interesujące są klasy przekształceń przestrzeni (i obiektów w danym problemie uwzględnianych) dokonywanych tak, aby zachować prawdziwość twierdzenia.

Teraz bardziej formalnie: punkt należący do E^d można interpretować jako wektor o d współrzędnych $(x_1, x_2, \ldots, x_d)$ (stosuje się też bardziej zwarty zapis, $\mathbf{x}$). Rozważmy przekształcenie $T: E^d \to E^d$ odwzorowujące E^d w nią samą. Takie odwzorowania zawsze będziemy interpretować jako przesunięcie punktów względem ustalonego układu współrzędnych, a nie jako zmianę układu współrzędnych przy niezmienionym punkcie (zobacz na przykład: [Birkhoff-MacLane (1965)]). Szczególnie interesują nad odwzorowania liniowe, czyli odwzorowania, w wyniku których *nowe* współrzędne punktu są liniowo zależne od starych współrzędnych. Tak więc, jeśli punkt p ma współrzędne $(x_1, x_2, \ldots, x_d)$, to jego obraz w odwzorowaniu T, punkt p', ma współrzędne:

$$x_i' = \sum_{j=1}^{d} a_{ji} x_j + c_i \quad (i = 1,2,\ldots,d) \tag{1.3}$$

albo, w bardziej zwartej formie,

$$\mathbf{x}' = \mathbf{x}A + \mathbf{c} \tag{1.4}$$

gdzie $A = \|a_{ij}\|$ jest macierzą $d \times d$, c jest d-elementowym wektorem stałym, a wszystkie wektory zapisywane są w formie wiersza.

Równanie (1.4) to ogólna postać *przekształcenia afinicznego*. W intuicyjnym zrozumieniu przekształceń afinicznych pomóc może rozważenie osobno dwóch przypadków szczególnych: $A = I$ (macierz jednostkowa) oraz $c = 0$. Kiedy $A = I$, równanie (1.4) przybiera postać:

$$\mathbf{x}' = \mathbf{x} + \mathbf{c} \tag{1.5}$$

i opisuje przekształcenie, w którym każdy punkt jest przesunięty o stały wektor c. Takie przekształcenia nazywamy po prostu *przesunięciami*. Kiedy z kolei c = 0, mamy:

$$\mathbf{x}' = \mathbf{x}A$$

czyli liniowe przekształcenie przestrzeni, odwzorowujące początek układu współrzędnych na samego siebie (czyli początek układu współrzędnych jest *punktem stałym* tego przekształcenia). W przypadku ogólnym na d-wymiarowe przekształcenie afiniczne można spojrzeć jako na $(d+1)$-wymiarowe przekształcenie liniowe w jednolitych współrzędnych, przy czym wektor punktu $(x_1, \ldots, x_d)$ jest rozszerzany o dodatkową składową $x_{d+1} = 1$. Wobec tego równanie (1.4) można zapisać jako:

[5] Czyli taki, w którym osie będą parami prostopadłe, zaś odcinki jednostkowe na nich odłożone będą miały takie same długości.

$$(\mathbf{x}', 1) = (\mathbf{x}, 1)\begin{bmatrix} A & 0 \\ \mathbf{c} & 1 \end{bmatrix} \tag{1.6}$$

Ważny podział przekształceń afinicznych oparty jest na właściwościach macierzy A. W szczególności różne gałęzie geometrii zajmować się mogą różnymi właściwościami, *niezmiennymi* przy rozmaitych przekształceniach. Ta bardzo ważna uwaga została uczyniona przez Kleina ponad wiek temu [Klein (1872)] i stała się teraz jednym z pryncypiów nauczania geometrii.

Zaczniemy od rozważenia przypadku, w którym A jest dowolną macierzą odwracalną. Oczywista, że takie przekształcenia tworzą grupę, co można łatwo udowodnić[6]. Grupę taką nazywamy *grupą afiniczną*, zaś *geometria afiniczna* zajmuje się właściwościami zachowywanymi przez przekształcenia takiej grupy (czyli *niezmiennikami*). Podstawowy niezmiennik geometrii afinicznej to padanie, czyli przynależność punktu p do prostej l.

Rozważmy następnie przypadek szczególny, zdefiniowany tożsamością:

$$AA^T = \lambda^2 I \tag{1.7}$$

gdzie λ jest stałą rzeczywistą (indeks górny T oznacza transpozycję macierzy). Taka właściwość charakteryzuje ważną podgrupę grupy afinicznej, znaną jako *grupa podobieństwa*. Jeśli zachowany jest warunek (1.7), łatwo sprawdzić, że stosunek odległości między punktami jest zachowany (tak samo kąty i prostopadłość). Rzeczywiście: bez straty dla ogólności możemy rozważyć przypadek, kiedy w (1.4) $\mathbf{c} = 0$. Norma (kwadrat długości) wektora $\mathbf{x}$ (odcinek o jednym końcu w początku układu współrzędnych) wyliczana jest jako $\mathbf{x}\mathbf{x}^T$, więc norma obrazu $\mathbf{x}'$ w przypadku przekształcenia z grupy podobieństwa to:

$$\mathbf{x}'\mathbf{x}'^T = \mathbf{x}A(\mathbf{x}A)^T = \mathbf{x}AA^T\mathbf{x}^T = \mathbf{x}\lambda^2 I\mathbf{x}^T = \lambda^2\mathbf{x}\mathbf{x}^T$$

skąd wynika, że wektor $\mathbf{x}$ uległ wydłużeniu o $\pm\lambda$. Następnie, jeśli mamy dwa wektory $\mathbf{x}$ i $\mathbf{y}$, mnożąc ich obrazy otrzymujemy:

$$\mathbf{x}'\mathbf{y}'^T = \lambda^2\mathbf{x}\mathbf{y}^T$$

Skoro $\mathbf{x}'\mathbf{y}'^T = |\mathbf{x}'|\cdot|\mathbf{y}'|\cos(\mathbf{x}'\mathbf{y}')$ i $\mathbf{x}\mathbf{y}^T = |\mathbf{x}|\cdot|\mathbf{y}|\cos(\mathbf{x},\mathbf{y})$, to

$$\cos(\mathbf{x}', \mathbf{y}') = \cos(\mathbf{x}, \mathbf{y})$$

co chcieliśmy udowodnić.

[6] Domkniętość, łączność, istnienie identyczności i odwrotności to bezpośrednie konsekwencje takich samych właściwości grupy niejednostkowych macierzy $d\times d$.

Teraz przyjrzyjmy się zawężeniu grupy afinicznej za pomocą warunku:

$$|A| = \pm 1$$

gdzie $|A|$ to wyznacznik A. Podgrupa przekształceń o takiej właściwości to grupa *niezmienniczo afiniczna*. Niezmiennikiem tej grupy jest *objętość*. Faktycznie, jeśli mamy zbiór d wektorów $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_d$, wartość bezwzględna wyznacznika macierzy $[\mathbf{x}_1^T ... \mathbf{x}_d^T]$ to objętość hiper-równoległościanu, wyznaczonego podanymi wektorami. Jeśli weźmiemy pod uwagę obrazy tych wektorów, $\mathbf{x}'_i = \mathbf{x}_i A$, $i = 1,...,d$, mamy $\left|[\mathbf{x}'^T_1 ...\mathbf{x}'^T_d]\right| = |A|\ \left|[\mathbf{x}^T_1 ...\mathbf{x}^T_d]\right| = \pm\left|[\mathbf{x}^T_1 ...\mathbf{x}^T_d]\right|$, co już wystarcza do wywnioskowania niezmienniczości.

Jeśli weźmiemy teraz pod uwagę przecięcie obu powyższych grup, otrzymamy nową grupę przekształceń, grupę *ortogonalną*. Niezmiennikiem tej grupy jest *odległość*. Wyznacznik $A \cdot A^T$ to

$$\left|A \cdot A^T\right| = |A| \cdot \left|A^T\right| = |A|^2 = 1$$

ale zgodnie z (1.7) $\left|AA^t\right| = \lambda^2$, zatem $\lambda = \pm 1$. Odległość $d(\mathbf{x}, \mathbf{y})$ między punktami $\mathbf{x}$ i $\mathbf{y}$ to wartość bezwzględna kwadratowego pierwiastka ich różnicy, czyli:

$$\mathrm{d}(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y}) \cdot (\mathbf{x}-\mathbf{y})^T}$$

Wobec tego mamy:

$$d(\mathbf{x}',\mathbf{y}') = \sqrt{(\mathbf{x}'-\mathbf{y}') \cdot (\mathbf{x}'-\mathbf{y}')^T} = \sqrt{(\mathbf{x}-\mathbf{y})AA^T(\mathbf{x}-\mathbf{y})^T} = \sqrt{(\mathbf{x}-\mathbf{y}) \cdot (\mathbf{x}-\mathbf{y})^T} = d(\mathbf{x},\mathbf{y})$$

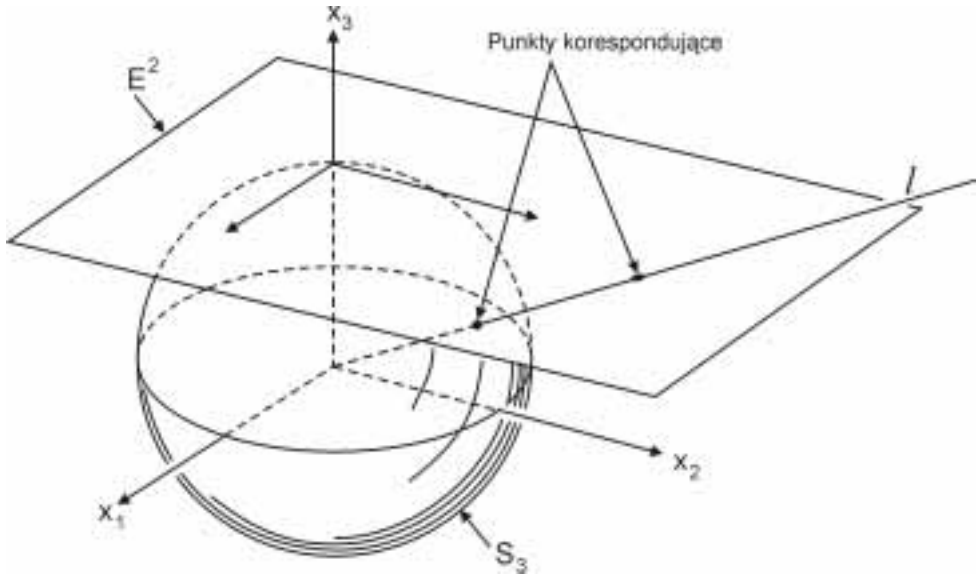
Przekształcenia afiniczne zachowujące odległości (przeto i pole, kąty i prostopadłość) to *ruchy sztywne*; stanowią one podstawę geometrii euklidesowej.

Teraz wróćmy do równania (1.6) opisującego konkretne przekształcenie (d+1)-wymiarowej przestrzeni wektorowej i zrezygnujmy z ograniczeń na ostatnią kolumnę macierzy przekształcenia. Otrzymamy związek:

$$\xi' = \xi B \tag{1.8}$$

gdzie ξ' i ξ to wektory (d+1)-wymiarowej przestrzeni euklidesowej V_{d+1}, zaś B to macierz (d+1)×(d+1), co do której zakładamy, że jest odwracalna.

Ograniczmy naszą uwagę do kierunku wektora, pomijając jego długość — dwa współliniowe wektory ξ i $c\xi$ ($c \neq 0$) są uważane za równoważne. Jako reprezentanta takiej klasy równoważności wybrać możemy punkt, w którym prosta $l = \{c\xi: c \in \mathrm{R}\}$ przebija sferę jednostkową S^{d+1} w przestrzeni E^{d+1} (zauważmy, że S^{d+1} jest rozmaitością d-wymiarową, czyli jej punkty można zidentyfikować za pomocą d parametrów). Jeśli dobierzemy taką wartość c, dla której ostatni składnik $c\xi$ jest równy 1, otrzymamy punkt, w którym prosta l przebija hiperpłaszczyznę $x_{d+1} = 1$ (rysunek 1.4 pokazuje tę sytuację dla $d = 2$). Mamy zatem wzajemną odpowiedniość punktów płaszczyzny $x_{d+1} = 1$ i punktów spełniającej



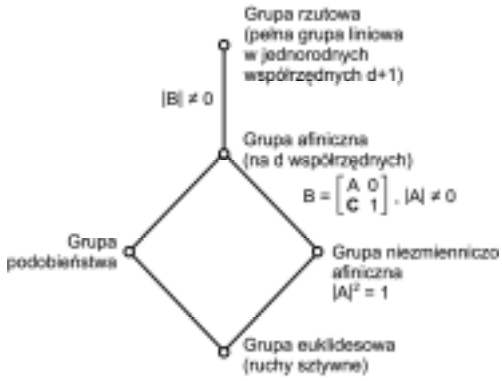
RYSUNEK 1.4.
Wzajemna odpowiedniość współrzędnych jednorodnych i niejednorodnych (zwykłych)

warunek $x_{d+1} > 0$ półsfery S^{d+1}. Taką wzajemną odpowiedniość nazywamy *rzutowaniem ośrodkowym* (o środku w początku układu współrzędnych E^{d+1}). Zauważmy, że hiperpłaszczyzna $x_{d+1} = 1$ sama jest przestrzenią E^d o współrzędnych $x_1, \ldots, x_d$.

Tak więc zinterpretowaliśmy wektor $(\xi_1, \ldots, \xi_d, \xi_{d+1})$ $(\xi_{d+1} \neq 0)$, przyłożony do początku układu współrzędnych E^{d+1}, jako punkt $(x_1, \ldots, x_d)$ przestrzeni E^d taki, że $x_j = \xi_j / \xi_{d+1}$. Jeśli zapiszemy punkt za pomocą $(d+1)$ składowych wektora, otrzymamy klasyczny zapis punktu we współrzędnych jednorodnych, które umożliwiają zapisanie punktów w nieskończoności przy zgodzie na $\xi_{d+1} = 0$.

Wracając do rozważań dotyczących operacji na grupie przekształceń, jeśli w (1.8) zgodzimy się, aby B miało postać $B = \begin{bmatrix} A & 0 \\ c & 1 \end{bmatrix}$, otrzymamy nową interpretację grup afinicznych E^d. Aby ułatwić dalszą dyskusję, przyjmijmy $d = 2$. Zauważmy, że rzutowanie ośrodkowe S^3 na płaszczyznę $x_3 = 1$ ustala przekształcenie odwracalne między wielkimi kołami S^3 a prostymi w E^2; wyjątkiem jest wielkie koło na „równiku" (płaszczyzna $x_3 = 0$), które jest odwzorowywane na nieskończoność E^2. Jednak każde przekształcenie (1.8) odwzorowuje płaszczyznę ze środkiem na płaszczyznę ze środkiem, a w szczególności płaszczyzna „równikowa" może być odwzorowana na dowolną inną płaszczyznę. Jeśli zinterpretujemy to przy rzutowaniu ośrodkowym na $x_3 = 1$, prostą w nieskończoności można odwzorować na dowolną prostą w skończoności. Ta swoboda wyboru umożliwia odwzorowanie wszystkich krzywych stożkowych (elipsy, paraboli i hiperboli) na okrąg, zatem w grupie przekształceń (1.8) wszystkie krzywe stożkowe są równoważne. Jest to *grupa rzutowa*.

Cały powyższy opis podsumować można w diagramie Hassego [Birkhoff-MacLane (1965)], opartym na relacji „podgrupa" i pokazanym na rysunku 1.5.



RYSUNEK 1.5. Zawieranie się w sobie grup przekształceń liniowych (diagram Hassego)

1.3.3. Dualność geometryczna. Biegunowość

Rozważmy teraz alternatywną interpretację przekształcenia (1.8), znów odwołując się do konkretnego przypadku, gdy $d = 2$. Widzieliśmy już, że (1.8) odwzorowuje punkty i proste na proste w E^2, czyli zachowuje wymiary przekształcanych obiektów. Zakładając, że $|B| \neq 0$, zapiszmy (1.8) jako:

$$\boldsymbol{\eta} = \xi B \tag{1.9}$$

wobec czego wektory oznaczone jako ξ interpretujemy jako kierunki (czyli proste przechodzące przez początek układu współrzędnych), a wektory oznaczane przez $\boldsymbol{\eta}$ reprezentują kierunki prostopadłe do płaszczyzny przechodzącej przez środek układu. Zależność (1.9) interpretuje się jako przekształcenie prostej (oznaczonej przez ξ) w E^3 na płaszczyznę (oznaczoną przez $\boldsymbol{\eta}$); nazywamy ten związek korelacją. Jest to przypadek szczególny dla $d = 2$ — ogólnej właściwości, wedle której w E^{d+1} zależność (1.9) odwzorowuje rozmaitość liniową wymiaru $s \leq d+1$ na rozmaitość liniową wymiaru $d+1-s$ do niej *dualną*: przy tej interpretacji (1.9) nazywamy *przekształceniem dualnym* i do opisu związku płaszczyzn na proste możemy użyć tej samej macierzy B. Jeśli mamy korelację prostej na płaszczyznę opisaną macierzą B, szukamy korelacji płaszczyzny na prostą opisanej macierzą D taką, aby para (B, D) zachowywała *przynależność*, to znaczy jeśli prosta ξ należy do płaszczyzny $\boldsymbol{\eta}$, to prosta $\boldsymbol{\eta}D$ należy do płaszczyzny ξB. Oczywiście prosta ξ należy do płaszczyzny $\boldsymbol{\eta}$ wtedy i tylko wtedy, gdy

$$\xi \cdot \boldsymbol{\eta}^T = 0$$

Analogicznie żądamy, aby:

$$\xi B(\boldsymbol{\eta}D)^T = \xi BD^T\boldsymbol{\eta}^T = 0$$

Jako że ostatnia zależność obowiązuje dla dowolnie wybranych ξ i $\boldsymbol{\eta}$, otrzymujemy tożsamość $BD^T = kI$, czyli:

$$D = k(B^{-1})^T$$

dla pewnej stałej k.

Zauważmy, że iloczyn BD przekształca proste i płaszczyzny na płaszczyzny. Jeśli mamy zachowującą przynależność parę $(B,(B^{-1})^T)$, możemy zażądać, aby iloczyn $B \cdot (B^{-1})^T$ przekształcał każdą prostą na nią samą i każdą płaszczyznę na nią samą, czyli żądamy, aby $(B(B^{-1})^T) = kI$, co jest równoważne $B = kB^T$. W przypadku tej ostatniej tożsamości musimy mieć $k = \pm 1$. Szczególnie interesującym przypadkiem jest

$$B = B^T \tag{1.10}$$

czyli kiedy B jest odwracalną macierzą symetryczną 3×3.

Zastanówmy się teraz nad działaniem korelacji opisanej zależnością (1.9) w rzutowaniu ośrodkowym na płaszczyznę $x_3 = 1$. Wiemy, że punkty są przekształcane na proste, a proste na punkty; w dalszej części tego punktu zbadamy to przekształcenie w E^2, przez ξ oznaczać będziemy punkt, a przez $\boldsymbol{\eta}$ prostą. Jeśli macierz B o wymiarach 3×3 spełnia warunek $B = B^T$, jeśli $\mathbf{x} = (x_1, x_2, x_3)$, wiadomo [Birkhoff-MacLane (1965)], że:

$$\mathbf{x}B\mathbf{x}^T = 0$$

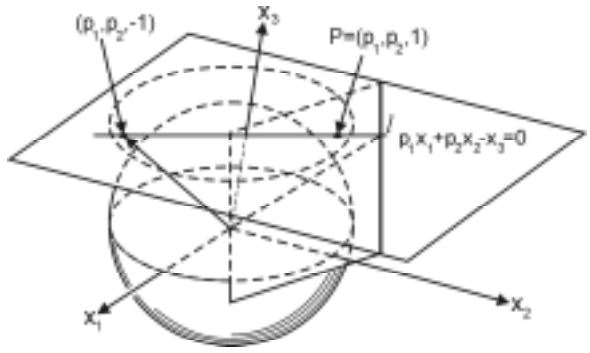
jest równaniem stożkowej, zapisanym w jednorodnych współrzędnych płaszczyzny (mówimy tu o *stożkowej określonej przez B*). Przyjmijmy teraz stałe ξ (interpretowane jako punkt płaszczyzny, zapisany we współrzędnych jednorodnych) takie, że $\xi B\xi^T = 0$. Z definicji punkt ξ leży na stożkowej zdefiniowanej przez BW. Jeśli nazwiemy punkt ξ *biegunem*, zaś prostą $\xi B\mathbf{x}^T = 0$ *biegunową* ξ, widzimy, że biegun na stożkowej należy do swojej własnej biegunowej. Takie przekształcenie punktów na proste i z powrotem nazywamy *przekształceniem biegunowym*; jest ono przydatne przy tworzeniu algorytmów geometrycznych. Nasza intuicja lepiej pozwala sobie wyobrażać punkty niż proste (w E^2) czy płaszczyzny (w E^3), więc z opisywanego przekształcenia będziemy korzystać w dalszych rozdziałach.

W szczególności możemy wybrać jako B macierz opisującą okrąg jednostkowy w płaszczyźnie E^2 (zobacz rysunek 1.6). W takim wypadku

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

czyli punkt $\mathrm{p} = (p_1, p_2, 1)$ jest odwzorowywany na prostą l, której równanie to $p_1x_1+p_2x_2-x_3 = 0$ (we współrzędnych jednorodnych). Odległość p od początku układu współrzędnych to $\sqrt{\mathrm{p}_1{}^2 + \mathrm{p}_2{}^2}$, zaś odległość l od początku układu współrzędnych to $1/\sqrt{\mathrm{p}_1{}^2 + \mathrm{p}_2{}^2}$. Tak więc widać, że w przypadku tego szczególnego przekształcenia biegunowego:

$$\text{odległość}(\mathrm{p}, 0) \times \text{odległość}(\text{biegunowa}(\mathrm{p}), 0) = 1$$



RYSUNEK 1.6. Ilustracja zależności bieguna i biegunowej przy przekształceniu biegunowym względem okręgu jednostkowego

Zauważmy też, że w przypadku tego przekształcenia zbędne są jakiekolwiek obliczenia, a wystarczy inaczej zinterpretować trójkę (p_1, p_2, p_3): albo jako współrzędne (p_1/p_3, p_2/p_3) punktu, albo jako współczynniki prostej o równaniu $p_1x_1+p_2x_2-p_3x_3 = 0$ we współrzędnych jednorodnych.

W dalszej części tego rozdziału przyjrzymy się ciekawym zastosowaniom tej dualności geometrycznej.

1.4. Modele obliczeniowe

Podstawowe pytanie metodologiczne, które trzeba postawić przed zajęciem się analizą algorytmów, to dokładne opisanie przyjętego *modelu obliczeniowego*. Algorytm, który ma służyć do rozwiązania jakiegoś problemu, musi być przede wszystkim oceniony pod względem kosztów jako funkcji wielkości problemu. Zasadnicze znaczenie przyjętego modelu wynika z następującego spostrzeżenia:

> *Model obliczeniowy opisuje elementarne operacje, które można wykonać, oraz podaje koszt ich wykonania.*

Operacje elementarne to te operacje, którym przypisujemy *stały* koszt, aczkolwiek koszt ten może być różny dla różnych operacji. Przykładowo, jeśli operacje elementarne dotyczą poszczególnych cyfr składających się na liczby (jak funkcje logiczne na dwóch zmiennych binarnych, opisywane w modelu maszyny Turinga), to koszt dodawania dwóch liczb całkowitych rośnie wraz z długością argumentów, a jest stały w przypadku modeli, w których argumenty mają stałą długość (stała długość argumentów jest cechą charakterystyczną wszystkich używanych dzisiaj komputerów). Wybierając jakiś model, zwykle jesteśmy zmuszeni do kompromisów między realizmem a możliwością przeprowadzenia wyliczeń; wybieramy taki sposób działania, który zapewni wykorzystanie podstawowych cech używanych komputerów, ale będzie dość prosty, aby przeprowadzić gruntowną analizę.

Jaki model będzie odpowiedni w przypadku zastosowań geometrycznych? Aby odpowiedzieć sobie na to fundamentalne pytanie, musimy starannie przebadać naturę problemów, które chcemy rozwiązywać.

Jak wspomnieliśmy w podrozdziale 1.3, punkty w E^d to podstawowe obiekty geometrii obliczeniowej. Wprawdzie na punkt patrzymy jak na wektor we współrzędnych kartezjańskich, ale należy dążyć do tego, aby wybór układu współrzędnych nie wpływał znacząco na czas wykonywania algorytmu. Oznacza to, że model obliczeniowy musi pozwalać przekształcać w razie potrzeby układ odniesienia z kosztem stałym dla każdego punktu; koszt ten może zależeć od liczby wymiarów, ale nie może zależeć od liczby punktów.

Innymi słowy, zbiór N punktów w d wymiarach można zapisać względem wybranego kartezjańskiego układu odniesienia w czasie proporcjonalnym do N (przypomnijmy, że tego samego rzędu jest czas wczytywania punktów jako danych wejściowych), wobec czego możemy założyć, że punkty są dane od razu w wybranym układzie odniesienia.

Problemy występujące w geometrii obliczeniowej mają różnoraką naturę. Warto je na nasze potrzeby pogrupować w kilka kategorii:

(1) *Wybór podzbioru*. W tego typu problemach mamy zestaw obiektów i musimy wybrać podzbiór spełniający pewne narzucone kryteria. Przykładem może być znajdowanie dwóch najbliższych sobie punktów ze zbioru N punktów czy znajdowanie wierzchołków wielokąta wypukłego, rozpiętego na zbiorze. Podstawową cechą wybierania podzbioru jest to, że nie są tworzone żadne nowe zbiory; rozwiązanie składa się wyłącznie z elementów stanowiących dane wejściowe.

(2) *Wyliczanie*. Dany jest zbiór obiektów i trzeba wyliczyć wartość pewnego parametru geometrycznego tego zbioru. Operacje elementarne przyjętego modelu muszą być dostatecznie uniwersalne, aby umożliwić takie wyliczenie. Załóżmy na przykład, że pracujemy ze zbiorem punktów mających współrzędne wyrażone liczbami całkowitymi. Aby określić odległość między parą punktów, nie tylko musimy mieć możliwość zapisywania liczb niewymiernych, ale także wyliczania pierwiastków kwadratowych. W przypadku innych problemów konieczne może być użycie funkcji geometrycznych.

(3) *Decyzja*. Z dwiema powyższymi klasami w naturalny sposób wiąże się jeszcze klasa określana jako decyzja. A konkretnie:
   (i) Jeśli problem obliczeniowy $\mathtt{A}$ wymaga wartości parametru A, związany z problemem obliczeniowym problem decyzyjny $D(\mathtt{A})$ wymaga odpowiedzi TAK lub NIE na pytanie typu: „Czy $A \geq A_0$?", gdzie A_0 jest stałą.
   (ii) Jeśli w związku z problemem wyboru $\mathtt{A}$ niezbędny jest podzbiór danego zbioru S, spełniający pewien warunek P, $D(\mathtt{A})$ wymaga odpowiedzi TAK lub NIE na pytanie typu „Czy zbiór S' spełnia P?", gdzie S' jest podzbiorem zbioru S.

Szybko zauważymy, że musimy umieć posługiwać się liczbami *rzeczywistymi* (a nie tylko całkowitymi) oraz praktycznymi zaokrągleniami tych liczb. Tworząc model obliczeniowy, możemy zająć się abstrakcją, nie troszcząc się o zaokrąglenia związane z przyjętą reprezentacją liczb rzeczywistych. Skorzystamy z *urządzenia swobodnego dostępu* (oznaczanego jako RAM), podobnego do opisanego w [Aho-Hopcroft-Ullman (74)], ale takiego, *że w każdej jednostce zapisać można pojedynczą liczbę rzeczywistą. Oto operacje, które są elementarne i z których każda wykonuje się po koszcie jednostkowym*:

(1) Działania arytmetyczne (+, –, ×, /).
(2) Porównanie dwóch liczb rzeczywistych ($<, \leq, =, \neq, \geq, >$).
(3) Pośrednie adresowanie pamięci (jedynie adresy całkowitoliczbowe).

Opcjonalnie, w niektórych aplikacjach:

(4) Pierwiastkowanie k-tego stopnia, funkcje trygonometryczne, EXP, LOG (ogólnie rzecz biorąc, funkcje analityczne).

Model ten będziemy określać mianem *rzeczywistego RAM*. Ściśle odpowiada to programom, jakie zwykle pisze się w językach algorytmicznych wysokiego poziomu, jak FORTRAN i ALGOL, gdzie zwykle zmienne typu REAL traktuje się jako mające nieograniczoną dokładność. Na naszym poziomie abstrakcji możemy pominąć pytania typu „jak liczbę rzeczywistą można odczytać lub zapisać w skończonym czasie".

Ustalenie dolnych ograniczeń szybkości rozwiązywania danego problemu jest jednym z fundamentalnych zadań analizy algorytmów, gdyż jest to miara wydajności algorytmu. W przypadku ogólnym jest to bardzo trudne zadanie. Czasami można powiązać trudność problemu z trudnością innego problemu o znanej złożoności; używa się w tym celu techniki *przekształcania problemów*[7]. Załóżmy, że mamy dwa problemy, A i B, które są ze sobą tak powiązane, że problem A można rozwiązać następująco:

(1) Dane wejściowe problemu A są przekształcane w postać danych wejściowych problemu B.
(2) Rozwiązywany jest problem B.
(3) Wyniki problemu B są przekształcane w poprawne rozwiązanie problemu A.

Mówimy wtedy, że problem A *został przekształcony* w problem B[8]. Jeśli powyższe kroki przekształcenia — 1. i 3. — mogą być wykonane w czasie $O(\tau(N))$, gdzie N jest tradycyjnie „rozmiarem" problemu A, to mówimy, że A jest $\tau(N)$-przekształcalne w B, co zapisujemy krótko:

$$A \propto_{\tau(N)} B$$

W przypadku ogólnym możliwość przekształcenia nie jest relacją symetryczną; w przypadku szczególnym, kiedy A i B dają się w siebie nawzajem przekształcić, mówimy, że są one sobie *równoważne*.

Następujące dwa stwierdzenia charakteryzują siłę techniki przekształcenia przy założeniu, że zachowany jest rząd wielkości problemu. Stwierdzenia te są oczywiste.

---

[7] Technikę tę bardzo często określa się jako *redukcję*. Słowo „redukcja" sugeruje przekształcanie w *prostszy* problem (co nie jest prawdą), więc tej nazwy nie będziemy używać.

[8] Możliwość przekształcenia (czyli redukowalność) zwykle definiuje się jako relację języków, kiedy nie jest konieczne żadne przekształcanie wyników, gdyż wynik z przyjmującego łańcuch jest zerem lub jedynką. W przypadku problemów geometrycznych potrzebujemy większej elastyczności, której osiągnięcie wymaga ogólniejszej definicji.

Stwierdzenie 1. (dolne ograniczenie przez możliwość przekształcenia). *Jeśli wiadomo, że problem* A *wymaga czasu* $T(N)$ *i* A *jest* $\tau(N)$*-przekształcalny w* B ($A \propto_{\tau(N)} B$), *to* B *wymaga co najmniej czasu* $T(N)$–$O(\tau(N))$.

Stwierdzenie 2. (górne ograniczenie przez możliwość przekształcenia). *Jeśli problem* B *można rozwiązać w czasie* $T(N)$, *a* A *daje się* $\tau(N)$*-przekształcić w* B ($A \propto_{\tau(N)} B$), *to* B *wymaga co najwyżej czasu* $T(N)$+$O(\tau(N))$.

Sytuację tę przedstawiono obrazowo na rysunku 1.7, gdzie widać przeniesienie dolnego i górnego ograniczenia z jednego problemu na drugi. Przeniesienie to obowiązuje, o ile tylko $\tau(N) = O(T(N))$, czyli kiedy przekształcenie nie zajmuje znaczącej części czasu obliczeniowego.

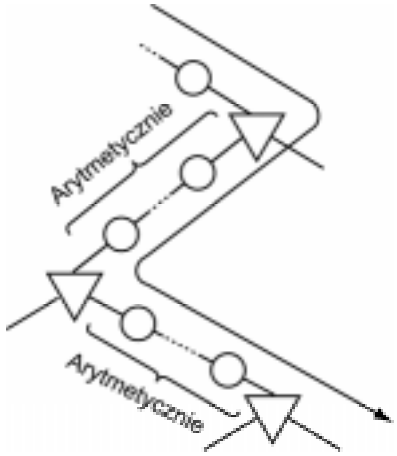
RYSUNEK 1.7. Przeniesienie górnych i dolnych ograniczeń między dającymi się przekształcić problemami

Odwołując się do naszej poprzedniej klasyfikacji problemów, rozpatrzmy pewien problem A (czy to obliczeniowy, czy związany z wybieraniem podzbioru) i związany z nim problem decyzyjny $D(A)$. Natychmiast widać, że $D(A) \propto_{O(N)} A$, gdyż:

(1) Jeśli A jest problemem obliczeniowym, żadne przekształcanie danych wejściowych nie jest konieczne (wobec czego krok 1. procedury przekształcającej jest pusty) i rozwiązanie A musi być porównane, w stałym czasie $O(1)$, ze stałą wartością wyznaczoną przez $D(A)$.
(2) Jeśli A jest problemem wyboru podzbioru, zbiór S' danych wejściowych $D(A)$ jest przekazywany jako dane wejściowe A (wobec czego krok 1. jest pusty), zatem rozwiązanie A jest sprawdzane w czasie $O(N)$, aby sprawdzić, czy jego liczebność jest zgodna z S'.

Jest to bardzo ważne spostrzeżenie, gdyż wskazuje ono, że jeśli chcemy obniżyć ograniczenie dolne, możemy ograniczyć nasz obszar zainteresowań do „problemów decyzyjnych".

Kiedy w „rzeczywistym RAM" wykonywany jest algorytm dotyczący problemu decyzji, jego zachowanie opisuje się jako ciąg operacji dwojakiego rodzaju: arytmetycznych i porównań. W tym ciągu porównania odgrywają kluczową rolę, gdyż — w zależności od wyniku — algorytm może się rozgałęziać w każdej operacji. Sytuacja ta jest obrazowo przedstawiona na rysunku 1.8, przy czym każdy węzeł oznaczony kółkiem to operacja arytmetyczna, zaś trójkąt to porównanie. Innymi słowy, obliczenia wykonywane przez RAM mogą być traktowane jako ścieżka w drzewie. Drzewo to zawiera opis wyjątkowo ważnego modelu obliczeniowego, nazywanego „algebraicznym drzewem decyzji". Teraz model ten sformalizujemy.



RYSUNEK 1.8. Wyliczenie jako ścieżka w drzewie decyzyjnym

Algebraiczne drzewo decyzji [Reingold (1972); Rabin (1972); Dobkin-Lipton (1979)] na zbiorze zmiennych $\{x_1, \ldots, x_n\}$ to program z instrukcjami $L_1, L_2, \ldots, L_p$ w postaci:

(1) L_r: Wylicz $f(x_1, \ldots, x_n)$; jeśli f: 0 — przejdź do L_i, w przeciwnym wypadku — przejdź do L_j (: oznacza dowolną relację porównania).
(2) L_s: Przerwij i zwróć wartość YES (TAK; zatwierdzone dane wejściowe problemu decyzyjnego).
(3) L_t: Przerwij i zwróć wartość NO (NIE; odrzucone dane wejściowe problemu decyzyjnego).

W kroku 1. f jest *funkcją algebraiczną* (wielomianem stopnia *degree*(f)). Zakłada się, że program nie ma pętli, czyli jego struktura to *drzewo* T takie, że węzeł nie będący liściem jest opisany przez

$$f_v(x_1, \ldots, x_n) : 0$$

gdzie f_v jest wielomianem na zmiennych $x_1, \ldots, x_n$, a: oznacza relację porównania. Zauważmy, że w tym opisie każda „ścieżka obliczeniowa" z rysunku 1.8 jest wciśnięta do następnego węzła porównania. Korzeń T reprezentuje krok początkowy obliczeń, jego liście to możliwe sposoby zakończenia wykonywania programu i możliwe odpowiedzi. Nie tracąc na ogólności, zakładamy, że drzewo T jest *binarne*[9].

Wprawdzie program algebraicznego drzewa decyzyjnego może być znacznie bardziej rozwlekły niż odpowiadający mu program RAM, ale oba programy zachowywać się będą *identycznie* przy rozwiązywaniu danej klasy problemów. W szczególności czas wykonania najgorszego możliwego przypadku programu RAM jest co najmniej proporcjonalny do długości najdłuższej ścieżki korzenia drzewa decyzji. Pokazuje to, jak istotny jest model drzewa decyzyjnego, gdyż struktura drzewiasta dobrze poddaje się poprawianiu ograniczeń przy analizie w głąb.

[9] Zauważmy, że stopień węzłów T jest wielokrotnością możliwych wyników porównań. Hipoteza drzewa binarnego oparta jest na fakcie, że k-krotne rozgałęzienie można rozwinąć na $(k–1)$ rozgałęzień dwukrotnych.

Algebraiczne drzewo decyzji jest d-tego rzędu, jeśli d jest największym stopniem wielomianów $f_v(x_1, \ldots, x_n)$ dla każdego węzła v drzewa T. Model drzewa decyzyjnego pierwszego rzędu, czyli *liniowego*, oparty jest na bardzo zaawansowanym narzędziu, pozwalającym wyznaczać dolne ograniczenie wielu problemów. W podrozdziałach 2.2, 4.1.3, 5.2 i 8.4 zajmiemy się tą problematyką szczegółowiej. Jednak liniowe drzewo decyzyjne nie jest pozbawione wad. Pierwszą z nich jest to, że zdarzyć się może, że wszystkie znane algorytmy rozwiązania danego problemu używają funkcji stopnia ≥ 2, wobec czego dolne ograniczenie oparte na modelu liniowego drzewa decyzyjnego nie obowiązuje. Drugą wadą jest to, że ograniczenia oparte na modelu liniowego drzewa decyzyjnego nie stosują się do nieznanych jeszcze algorytmów, korzystających z funkcji wyższego stopnia.

Wyjątkowo ważne wyniki w tej materii dla $d \geq 2$ korzystając z klasycznych pojęć rzeczywistej geometrii algebraicznej uzyskali Steele-Yao (1982) i Ben-Or (1983). Podejście to oparte jest na następującym pomyśle: niech $x_1, x_2, \ldots, x_n$ będą parametrami problemu decyzyjnego, na którego poszczególne zestawy wartości można patrzeć jako na punkty w n-wymiarowej przestrzeni euklidesowej E^n. Problem decyzyjny wskazuje zbiór punktów $W \subseteq E^n$ czyli udziela odpowiedzi YES wtedy i tylko wtedy, gdy $(x_1, x_2, \ldots, x_n) \in W$ (mówimy wtedy, że *drzewo decyzyjne T rozwiązuje problem przynależności do W*). Załóżmy, że na podstawie analizy samego problemu znamy liczbę $\#(W)$ rozłącznych spójnych składników W. Każde obliczenie odpowiada niepowtarzalnej ścieżce $v_1, v_2, \ldots, v_{l-1}, v_l$ w drzewie T, gdzie v_1 jest korzeniem, a v_l liściem. Z każdym węzłem v_j danej ścieżki powiązana jest funkcja $f_{vj}(x_1, \ldots, x_n)$ taka, że dla $j = 1, \ldots, l-1$ $(x_1, \ldots, x_n)$ spełnia ograniczenia typu:

$$f_{vj} = 0 \text{ lub } f_{vj} \geq 0 \text{ lub } f_{vj} > 0 \tag{1.11}$$

Aby wyrobić sobie jakąś intuicję, najpierw zastanówmy się nad specjalnym przypadkiem dla $d = 1$ (model liniowego drzewa decyzyjnego lub obliczeniowego), opracowanym przez Dobkina i Liptona (1979). Niech $W \subseteq E^n$ będzie zbiorem przynależności dla danego problemu decyzyjnego, a $\#(W)$ niech oznacza liczbę rozłącznych spójnych składowych. Niech T będzie (binarnym) liniowym drzewem decyzji, zawierającym algorytm A, sprawdzający przynależność do W. Z każdym liściem T powiązana jest dziedzina E^n, a każdy liść oznacza „akceptację" lub „odrzucenie". Niech $\{W_1, \ldots, W_p\}$ będą składowymi W, $\{l_1, \ldots, l_r\}$ zbiorem liści, a D_j dziedziną związaną z l_j. Liść l_j jest uznawany za

$$\begin{cases} \text{akceptacja}, & \text{gdy } D_j \subseteq W \\ \text{odrzucenie}, & \text{w przeciwnym wypadku} \end{cases}$$

Dolne ograniczenie dotyczące r uzyskujemy, wykazując, że $r \geq \#(W)$. Faktycznie, tworzymy funkcję Y: $\{W_1, W_2, \ldots, W_p\} \to \{1,2, \ldots, r\}$ jako $Y(W_i) = \min\{j: j \in \{1,2, \ldots, r\} \text{ oraz } D_j \cap W_i \neq \emptyset\}$. Załóżmy teraz, że jest przeciwnie — istnieją dwa rozłączne podzbiory W_i i W_j takie, że $Y(W_i) = Y(W_j) = l_h$. Jako że algorytm A rozwiązuje problem należenia punktu q do W_i, liść l_h to liść akceptacji. Z drugiej strony, z definicji Y, jeśli $Y(W_j) = l_h$, to $W_i \cap D_h$ jest niepusty. Niech zatem q' będzie punktem z $W_i \cap D_h$. Analogicznie, q'' jest punktem w $W_j \cap D_h$. T jest *liniowym* drzewem decyzji, więc obszar E^n odpowiadający D_h jest przecięciem półprzestrzeni, czyli obszarem *wypukłym*. Oznacza to, że dowolna wypukła kombinacja punktów D_h należy do D_h (punkt 1.3.1). Rozważmy teraz odcinek $\overline{q'q''}$ (rysunek 1.9). Odcinek ten

RYSUNEK 1.9. Ilustracja dowodu, że $W_i = W_j$

przecina przynajmniej dwie składowe W (na pewno W_i i W_j), a jako że składowe te są rozłączne, zawiera przynajmniej jeden punkt $q''' \notin W$. Jednak z wypukłości wynika, że cały $\overline{q'q''}$ należy do D_h, zatem należy doń też q''', który okazuje się leżeć we wnętrzu W, co jest już sprzecznością. Tak więc T musi mieć przynajmniej p liści. Jako że najpłytsze binarne drzewo o danej liczbie liści jest zrównoważone, głębokość T wynosi przynajmniej $\log_2 p = \log_2 \#(W)$. Podsumowaniem niech będzie następujące twierdzenie:

Twierdzenie 1.1. (Dobkin-Lipton). *Dowolny algorytm liniowego drzewa decyzyjnego rozwiązujący problem przynależności do $W \subseteq E^n$ musi mieć głębokość nie mniejszą niż $\log_2 \#(W)$, gdzie $\#(W)$ jest liczbą rozłącznych spójnych składowych W.*

Niestety, zastosowanie opisanej powyżej techniki ogranicza się do algorytmów liniowych drzew wyliczeniowych, gdyż opieramy się w dużej mierze na tym, że dziedzina E^n związana z liściem drzewa jest wypukła. Kiedy maksymalny stopień wielomianów f_v jest większy lub równy 2, z tej przydatnej cechy nie możemy już skorzystać i potrzebne jest subtelniejsze rozumowanie.

Intuicyjnie, kiedy używane są wielomiany wyższego stopnia, dziedzina związana z danym liściem drzewa decyzyjnego może składać się z *kilku* rozłącznych składowych W. Jeśli uda się ograniczyć liczbę komponentów przypisanych do liścia na podstawie głębokości zagnieżdżenia tego liścia, nietrudno będzie ograniczyć głębokość T.

Kluczem do rozwiązania tego trudnego problemu jest błyskotliwe zastosowanie [Steele-Yao (1982), Ben-Or (1983)] klasycznego wyniku geometrii algebraicznej udowodnionego niezależnie przez Milnora (1964) i Thoma (1965). Wynik ten ma następującą postać:

Niech $\boldsymbol{V}$ będzie zbiorem punktów (formalnie *rozmaitością algebraiczną*) w m-wymiarowej przestrzeni kartezjańskiej E^m, opisanym równaniami wielomianowymi:

$$g_1(x_1, \ldots, x_m) = 0, \ldots, g_p(x_1, \ldots, x_m) = 0 \tag{1.12}$$

Wtedy, jeśli stopień każdego wielomianu g_i ($i = 1, \ldots, p$) jest $\leq d$, to liczba $\#(\boldsymbol{V})$ rozłącznych spójnych składowych[10] $\boldsymbol{V}$ jest ograniczona następująco:

[10] Tak naprawdę Milnor i Thom udowodnili silniejsze twierdzenie: $d(2d-1)^{m-1}$ jest górnym ograniczeniem dotyczącym sumy liczb Betti V, zaś liczba $\#(V)$ składowych spójnych jest jedną z liczb Bettiego (zerową).

$$\#(V) \leq d(2d-1)^{m-1}$$

Niestety, V zdefiniowano za pomocą równań (zobacz (1.12)), zaś ograniczenia na ścieżkę w T zawierają zarówno równania, jak i nierówności. Trudność ta została usunięta przez Bena i Ora, którzy przekształcili zadanie tak, że spełnione zostały założenia twierdzenia Milnora-Thoma. Przekształcenie to teraz opiszemy.

Zdefiniujmy $U \subseteq E^n$ jako zbiór punktów spełniających następujące warunki ($\mathbf{x} = (x_1, \ldots, x_n)$):

$$\begin{cases} q_1(\mathbf{x}) = 0, \ldots, q_r(\mathbf{x}) = 0 \\ p_1(\mathbf{x}) = 0, \ldots, p_s(\mathbf{x}) > 0 \\ p_{s+1}(\mathbf{x}) \geq 0, \ldots, p_h(\mathbf{x}) \geq 0 \end{cases} \tag{1.13}$$

gdzie q_i oraz p_j to wielomiany, a $d = \max\{2, \deg(q_i), \deg(p_j)\}$. Zauważmy, że mamy trzy rodzaje ograniczeń: równania, nierówności ostre i nierówności nieostre. Niech $\#(U)$ oznacza liczbę spójnych składowych zbioru U.

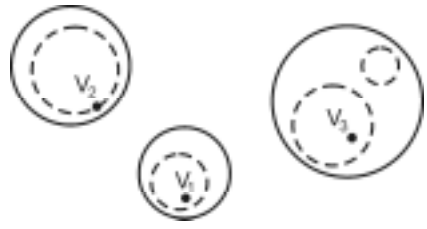
Pierwszy krok naszego przekształcenia polega na zastąpieniu ostrych nierówności nierównościami nieostrymi. Jako że $\#(U)$ ma wartość skończoną (zobacz [Milnor (1968)]), niech $\#(U)$ t; z każdej składowej U wybieramy punkt. Oznaczamy te punkty przez $\mathbf{v}_1, \ldots, \mathbf{v}_t$; niech:

$$\varepsilon = \min\{p_i(\mathbf{v}_j) : i = 1, \ldots, s; j = 1, \ldots, t\}$$

Jako że każdy $\mathbf{v}_j \in U$, $p_i(\mathbf{v}_j) > 0$ (dla $i = 1, \ldots, s$) oraz $\varepsilon > 0$. Oczywiście domknięty zbiór punktów U_ε zdefiniowany zależnościami:

$$\begin{cases} q_1(\mathbf{x}) = 0, \ldots, q_r(\mathbf{x}) = 0 \\ p_1(\mathbf{x}) \geq \varepsilon, \ldots, p_s(\mathbf{x}) \geq \varepsilon \\ p_{s+1}(\mathbf{x}) \geq 0, \ldots, p_h(\mathbf{x}) \geq 0 \end{cases} \tag{1.14}$$

jest zawarty w U, a $\#(U_\varepsilon) \geq \#(U)$, gdyż każdy $\mathbf{v}_j$ jest niezależną składową U_ε (zobacz rysunek 1.10).



RYSUNEK 1.10. Tworzenie U_ε dla U; samo U zaznaczono liniami ciągłymi, U_ε — przerywanymi

Drugi krok przekształcenia to wprowadzenie wolnej zmiennej y_j dla każdego $j = 1, \ldots, h$ (czyli dla każdej nierówności nieostrej), tak że na E^n można spojrzeć jako na podprzestrzeń E^{n+h}:

$$\begin{cases} q_1(\mathbf{x}) = 0, \ldots, q_r(\mathbf{x}) = 0 \\ p_1(\mathbf{x}) - \varepsilon - y_1^{\,2} = 0, \ \ldots, \ p_s(\mathbf{x}) - \varepsilon - y_s^{\,2} = 0 \\ p_{s+1}(\mathbf{x}) - y_{s+1}^{\,2} = 0, \ldots, p_h(\mathbf{x}) - y_h^{\,2} = 0 \end{cases} \tag{1.15}$$

Oczywiście zbiór U^* wszystkich rozwiązań (1.15) jest rozmaitością algebraiczną w E^{n+h}, zdefiniowaną wielomianami stopnia co najwyżej d, i spełnia warunki twierdzenia Milnora-Thoma. Wobec tego liczba $\#(U^*)$ składowych spójnych U^* jest ograniczona następująco:

$$\#(U^*) \le d(2d-1)^{n+h-1}$$

Zauważmy, U_ε to rzutowanie U^* na E^n. Jako że rzutowanie jest funkcją ciągłą, $\#(U_\varepsilon \le \#(U^*)$. Przypomnijmy, że $\#(U) \le \#(U_\varepsilon)$, wobec czego wnioskujemy, że:

$$\#(U) \le \#(U_\varepsilon) \le \#(U^*) \le d(2d-1)^{n+h-1}$$

co jest szukanym wynikiem. Faktycznie, jeśli (1.13) jest zbiorem warunków uzyskanych podczas przechodzenia po ścieżce w T od korzenia do liści, h — liczba nierówności — nie przekracza długości ścieżki. Wobec tego liść danej ścieżki jest powiązany z co najwyżej $d(2d-1)^{n+h-1}$ spójnymi składowymi zbioru wynikowego problemu W. Jeśli h^* jest głębokością T (długością najdłuższej ścieżki od korzenia do liścia), to T ma co najwyżej 2^{h^*} liści. Każdy liść odpowiada co najwyżej $d(2d-1)^{n+h-1}$ spójnych składowych W, wobec czego

$$\#(W) \le 2^{h^*} d(2d-1)^{n+h^*-1}$$

lub, równoważnie,

$$h^* \ge \frac{\log_2 \#(W)}{1+\log_2(2d-1)} - \frac{n\log_2(2d-1)}{1+\log_2(2d-1)} - \frac{\log_2 d - \log_2(2d+1)}{1+\log_2(2d-1)}. \tag{1.16}$$

Rozważania te podsumujmy w formie ważnego twierdzenia:

Twierdzenie 1.2.[11] *Niech W będzie zbiorem w przestrzeni kartezjańskiej E^n i niech T będzie algebraicznym drzewem decyzji ustalonego rzędu d ($d \ge 2$), które pozwala rozwiązać problem przynależności do W. Jeśli h^* jest głębokością T i $\#(W)$ jest liczbą rozłącznych spójnych składowych W, to $h^* = \Omega(\log\#(W) - n)$.*

Twierdzenie 1.2. uwzględnia też przypadek $d = 1$, gdyż dla dowolnego ustalonego $d \ge 2$ narzuca się ograniczenie dotyczące wielomianów niższego stopnia (przez ustawienie współczynników wyższych stopni na zero). Użycie hiperliniowych wielomianowych funkcji decyzyjnych nie zmienia zasadniczo natury problemu; po prostu skraca minimalną głębokość drzewa obliczeń przez stały mnożnik, zależny od maksymalnego stopnia d. Ostatnie twierdzenie to fundament wyznaczania dolnych ograniczeń, czym zajmiemy się w dalszych rozdziałach.

[11] Tak naprawdę Ben i Or udowodnili jeszcze lepszy wynik, niezależny od d.