Тема 28. Застосування дерев

28.1. Бінарне дерево пошуку

Бінарне дерево дуже зручний метод організації даних, у разі використання якого можна легко знайти будь-які конкретні дані чи виявити, що їх немає. Очевидно, що найнеефективніший спосіб пошуку – послідовний перегляд усіх даних. Справді, якщо потрібних даних немає, то для виявлення цього потрібно переглянути весь список. Бінарне дерево пошуку дає змогу уникнути цього. Єдина вимога – уведення для даних якогось лінійного порядку. Ним може бути, наприклад, алфавітний або числовий порядок. Лінійно впорядкувати можна теги, покажчики, файли чи інші ключі, які визначають дані. Але нас цікавитиме лише наявність якогось лінійного порядку.

У **бінарному дереві пошуку** кожній вершині присвоєно значення, яке називають **ключем**. Ключ – це елемент якоїсь лінійно впорядкованої множини: будь-які два її елементи можна порівняти (див. тему 4).

Під час побудови бінарного дерева пошуку використовують його рекурсивну властивість, яку можна описати наступним чином. Кожна вершина розбиває дерево на два піддерева. Ліве піддерево містить лише ключі, менші від ключа цієї вершини, а праве – ключі, більші від ключа вершини. Ця властивість повторюється для кожної вершини.

На рис. 28.1 зображений приклад такого дерева, де ключами є літери латинського алфавіту.

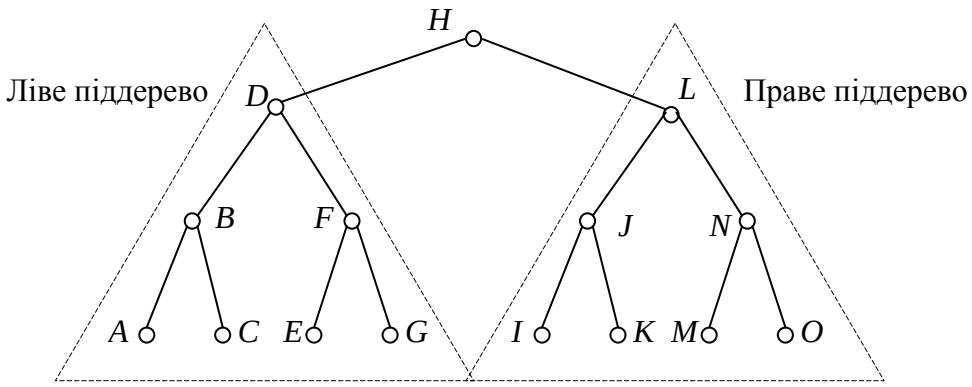

Рис. 28.1.

Розглянемо алгоритм додавання об’єкта до дерева пошуку, який будує бінарне дерево пошуку. Почнемо з дерева, що містить лише одну вершину. Означимо її як корінь. Перший об’єкт списку присвоюємо кореню; це ключ кореня. Щоб додати новий об’єкт виконуємо таку процедуру.

Алгоритм додавання об’єкта до дерева

1. Почати з кореня.
2. Якщо об’єкт менший, ніж ключ у вершині, то перейти до лівого сина.
3. Якщо об’єкт більший, ніж ключ у вершині, то перейти до правого сина.
4. Повторювати кроки 2 та 3, доки не досягнемо вершини, яку не визначено (тобто її немає).
5. Якщо досягнуто невизначену вершину, то визначити (тобто додати) вершину з новим об’єктом в якості ключа.

Побудуємо, наприклад, бінарне дерево пошуку для назв таких міст України: Київ, Львів, Харків, Донецьк, Івано-Франківськ, Дніпропетровськ, Одеса, Херсон, Чернігів, Сімферополь. Процес побудови зображено на рис. 28.2.

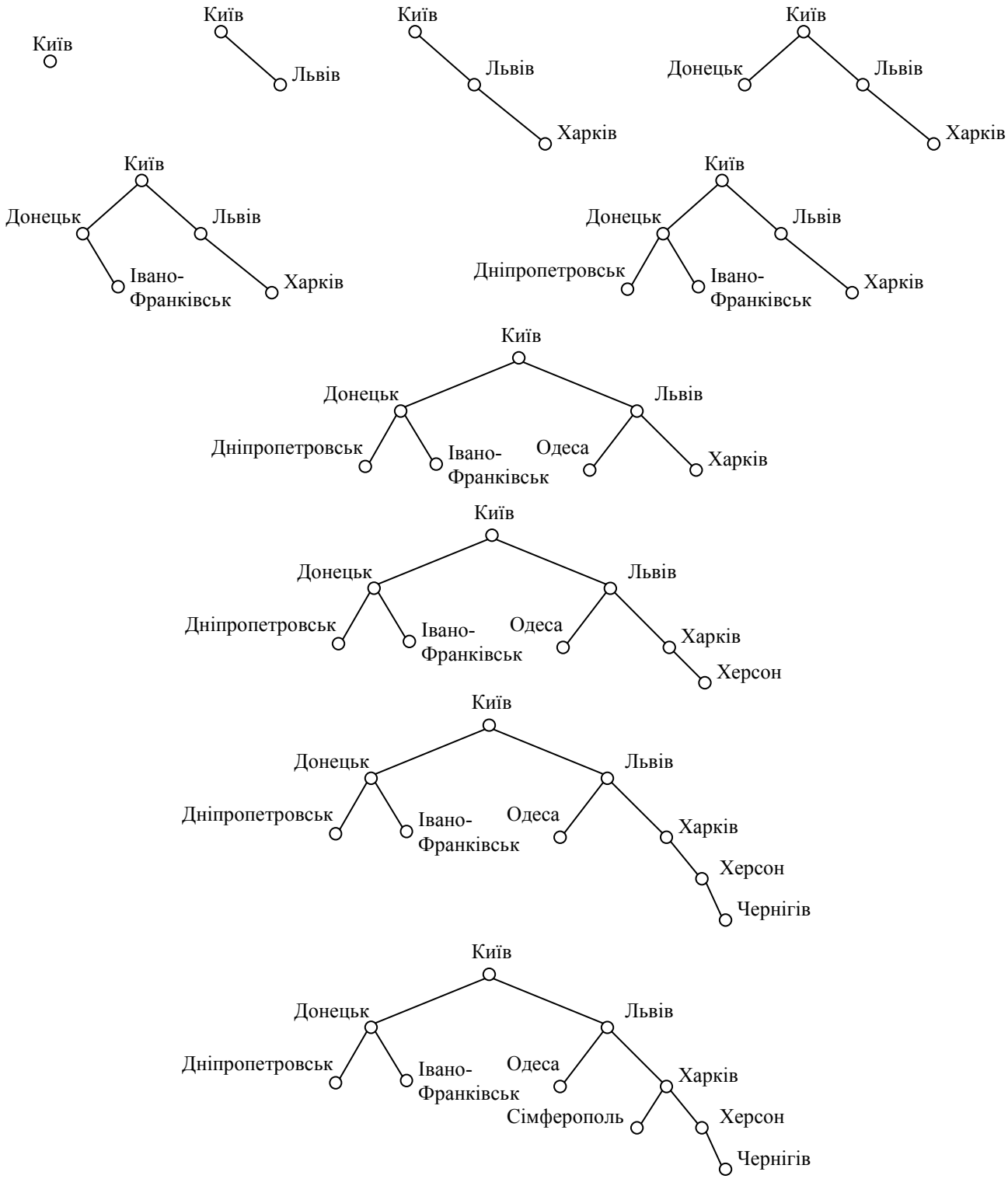

Рис. 28.2.

Оскільки метод побудови дерева пошуку описано, легко зрозуміти, як відбувається пошук елемента в дереві. Застосовують переважно той самий підхід. Окрім перевірки того, чи даний об’єкт більший або менший, ніж ключ у вершині, перевіряють також, чи збігається даний об’єкт із ключем у вершині. Якщо так, то процес пошуку завершено, якщо ні – описані дії повторюють. Якщо ж досягнуто вершину, яку не визначено, то це означає, що даний об’єкт не зберігається в дереві. Нижче наведено алгоритм пошуку об’єкту в бінарному дереві.

Алгоритм пошуку об’єкта в дереві

1. Почати з кореня.
2. Якщо об’єкт (його ключ) менший, ніж ключ у вершині, то перейти до лівого сина.

3. Якщо об’єкт (його ключ) більший, ніж ключ у вершині, то перейти до правого сина.
4. Якщо об’єкт дорівнює ключу у вершині, то об’єкт знайдено – кінець роботи.
5. Повторювати кроки 2, 3 та 4, доки не досягнемо вершини, яку не визначено.
6. Якщо досягнуто невизначену вершину, то даний об’єкт не зберігається в дереві – кінець роботи.

Оцінимо обчислювальну складність алгоритмів включення та пошуку (локалізації) об’єкта в бінарному дереві пошуку. Припустимо, що є бінарне дерево пошуку T для списку з n об’єктів. Із дерева T утворимо повне бінарне дерево U, для чого додамо нові вершини (без ключів) так, щоб кожна вершина з ключем мала двох синів. Це показано на рис. 28.3, де вершини без ключів позначено чорними кружечками.

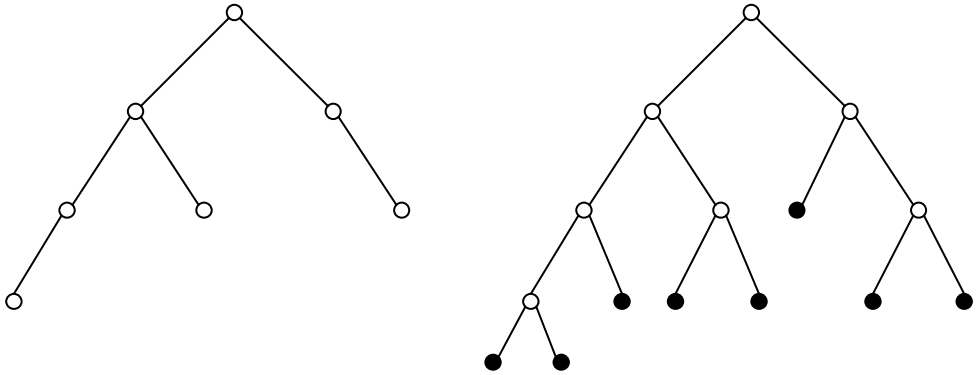

Рис. 28.3.

Коли це зроблено, можна легко локалізувати об’єкт або додати новий об’єкт як ключ без додавання вершини. Найбільша кількість порівнянь, потрібних для додавання нового об’єкту, дорівнює довжині найдовшого шляху в U від кореня до листка. Отже, дерево U має n внутрішніх вершин. За теоремою 27.2 кількість листків в ньому дорівнює $2n + 1 – n = n + 1$. Згідно з наслідком із теореми 27.3 висота дерева U більше чи дорівнює $\lceil log_2(n+1) \rceil$. Отже, потрібно щонайменше $\lceil log_2(n+1) \rceil$ порівнянь, щоб додати або знайти довільний об’єкт. Якщо дерево збалансоване, то його висота дорівнює $\lceil log_2(n+1) \rceil$. У цьому випадку для додавання чи знаходження об’єкту потрібно не більше $\lceil log_2(n+1) \rceil$ порівнянь.

Бінарне дерево пошуку може розбалансуватись унаслідок додавання нових об’єктів, тому потрібен алгоритм ребалансування (тобто відновлення збалансованості). Проте процедура додавання об’єкту, яка відновлює збалансоване дерево, навряд чи завжди доцільна, бо відновлення збалансованості дерева після випадкового додавання – досить складна операція.

Тому розглядають також збалансованість із дещо послабленими вимогами; зокрема, означають **АВЛ-дерево** – бінарне дерево, у якому висоти двох піддерев кожної з його вершин відрізняються не більше ніж на одиницю (рис. 28.4). Таке означення збалансованості широко використовують на практиці.

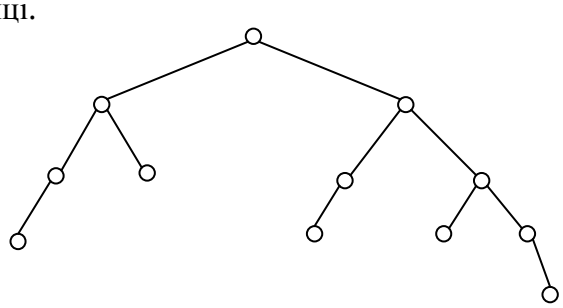

Рис. 28.4.

Для АВЛ-дерев є дуже ефективна процедура ребалансування. Водночас висота АВЛ-дерева незалежно від кількості вершин ніколи не перевищує висоту збалансованого дерева більше, ніж на 45%. Якщо позначити висоту АВЛ-дерева з n вершинами як $h(n)$, то

$$\log(n+1) \le h(n) < 1.44\log(n+2) - 0.328.$$

28.2. Бектрекінг (пошук із поверненнями)

Опишемо загальний метод, який дає змогу значно зменшити обсяг обчислювань в алгоритмах типу повного перебору всіх можливостей. Щоб застосувати цей метод, розв'язок задачі повинен мати вигляд скінченої послідовності (x_1, …, x_n). Головна ідея методу полягає в тому, що розв'язок будують поступово, починаючи з порожньої послідовності ∅ (довжиною 0). Загалом, якщо є частковий (неповний) розв'язок (x_1, …, x_i), де $i<n$, то намагаємося знайти таке допустиме значення x_{i+1}, що можна продовжувати (x_1, …, x_i, x_{i+1}) до отримання повного розв'язку. Якщо таке допустиме, але ще не використане значення x_{i+1} існує, то долучаємо цю нову компоненту до часткового розв'язку і продовжуємо процес для послідовності (x_1, …, x_i, x_{i+1}). Якщо такого значення x_{i+1} немає, то повертаємося до попередньої послідовності (x_1, …, x_{i-1}) і продовжуємо процес, шукаючи нове, ще не використане значення x_i'. Тому цей процес називають **бектрекінг** (англ. *backtracking* – пошук із поверненнями).

Роботу цього алгоритму можна інтерпретувати як процес обходу деякого дерева. Кожна його вершина відповідає деякій послідовності (x_1, …, x_i), причому вершини, які відповідають послідовностям вигляду (x_1, …, x_i, y), − сини цієї вершини. Корінь дерева відповідає порожній послідовності.

Виконується обхід цього дерева пошуком углиб. Окрім того, задають предикат P, означений на всіх вершинах дерева. Якщо $P(v)$ = False, то вершини піддерева з коренем у вершині v не розглядають, і обсяг перебору зменшується. Предикат $P(v)$ набуває значення False тоді, коли стає зрозумілим, що послідовність (x_1, …, x_i), яка відповідає вершині v, ніяким способом не можна добудувати до повного розв'язку.

Проілюструємо застосування алгоритму бектрекінгу на конкретних прикладах.

Задано множину натуральних чисел {x_1, …, x_n}. Потрібно знайти її підмножину, сума елементів якої дорівнює заданому числу M.

Починаємо з порожньої множини. Збільшуємо суму, послідовно добираючи доданки. Число з послідовності x_1, …, x_n долучають до суми, якщо сума після додавання цього числа не перевищує M. Якщо сума настільки велика, що додавання будь-якого нового числа перевищує M, то повертаємось і змінюємо останній доданок у сумі. На рис. 28.5 проілюстровано алгоритм бектрекінгу для задачі пошуку підмножини множини {31, 27, 15, 11, 7, 5} із сумою 39.

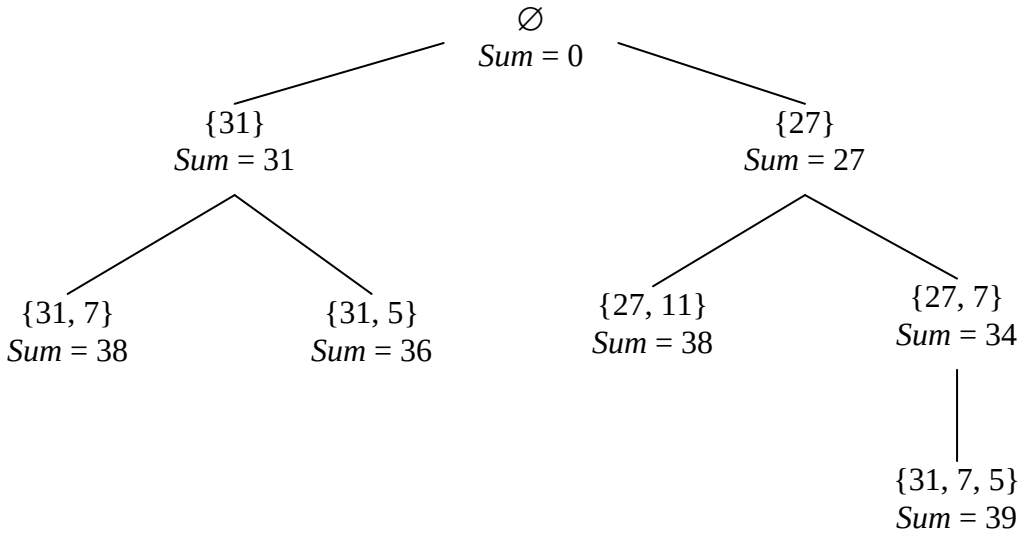

Рис. 28.5.

Маємо наступний приклад. Як n ферзів можна розмістити на шахівниці $n\times n$ так, щоб жодні два ферзі не били один одного?

Для розв'язання цієї задачі потрібно визначити n позицій на шахівниці так, щоб жодні дві позиції не були в одному рядку, в одному стовпці та на одній діагоналі. Діагональ містить усі позиції з координатами (i, j) такі, що $i + j = m$ для довільних m, або $i – j = m$ (тут i – номер рядка, j – номер стовпця). Починаємо з порожньої шахівниці. На $(k + 1)$-му кроці намагаємося розмістити нового ферзя в $(k + 1)$-му стовпці, причому на перших k стовпцях вже є ферзі. Перевіряємо клітинки в $(k + 1)$-му стовпці, починаючи з верхньої. Шукаємо таку позицію для ферзя, щоб він не був у рядку та на діагоналі з тими ферзями, які вже є на шахівниці. Якщо це неможливо, то повертаємося до місця ферзя на попередньому k-му кроці та розміщаємо цього ферзя на наступному можливому рядку в цьому k-му стовпці, якщо такий порядок є, а ні, то повертаємося до ферзя в $(k – 1)$-му стовпці. Алгоритм бектрекінгу для $n = 4$ наведений на рис. 28.6.

Кожній вершині дерева на рис. 28.6 відповідає послідовність довжиною від 0 до 4. Її k-й член дорівнює номеру клітинки з ферзем у k-му стовпці. Наприклад, вершинам шляху, який веде до розв'язку, відповідають такі послідовності: ∅, (2), (2, 4), (2, 4, 1), (2, 4, 1, 3).

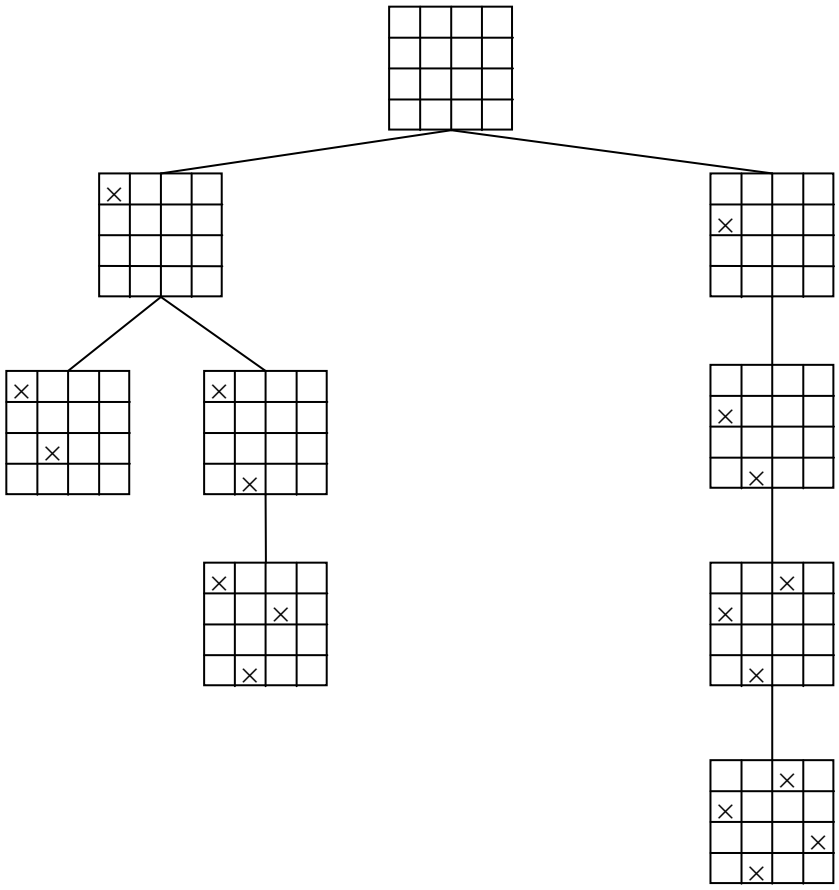

Рис. 28.6.

28.3. Дерево прийняття рішень

Кореневі дерева можна застосовувати для розв'язування задач прийняття рішень. Для цього використовують **дерева прийняття рішень**. Кожному листку такого дерева поставлено у відповідність рішення зі скінченної множини відомих рішень певної задачі, а кожній внутрішній вершині v – перевірку умови $P(v)$. Прийняття рішень за допомогою такого дерева полягає в побудові простого шляху від кореневої вершини до листка. Під час побудови шляху виконують перевірку умов у внутрішніх вершинах дерева. Значення умови $P(v)$ у вершині v задає ребро, яке буде додано в шлях після вершини v. Кінцева вершина побудованого шляху відповідає прийнятому рішенню.

Розглянемо таку відому логічну задачу. Серед дев'яти монет є одна фальшива: вона має іншу вагу ніж решта монет. До того ж, невідомо, чи є ця вага більшою, чи меншою. Потрібно знайти цю монету, використовуючи балансові терези, не більше ніж за три кроки.

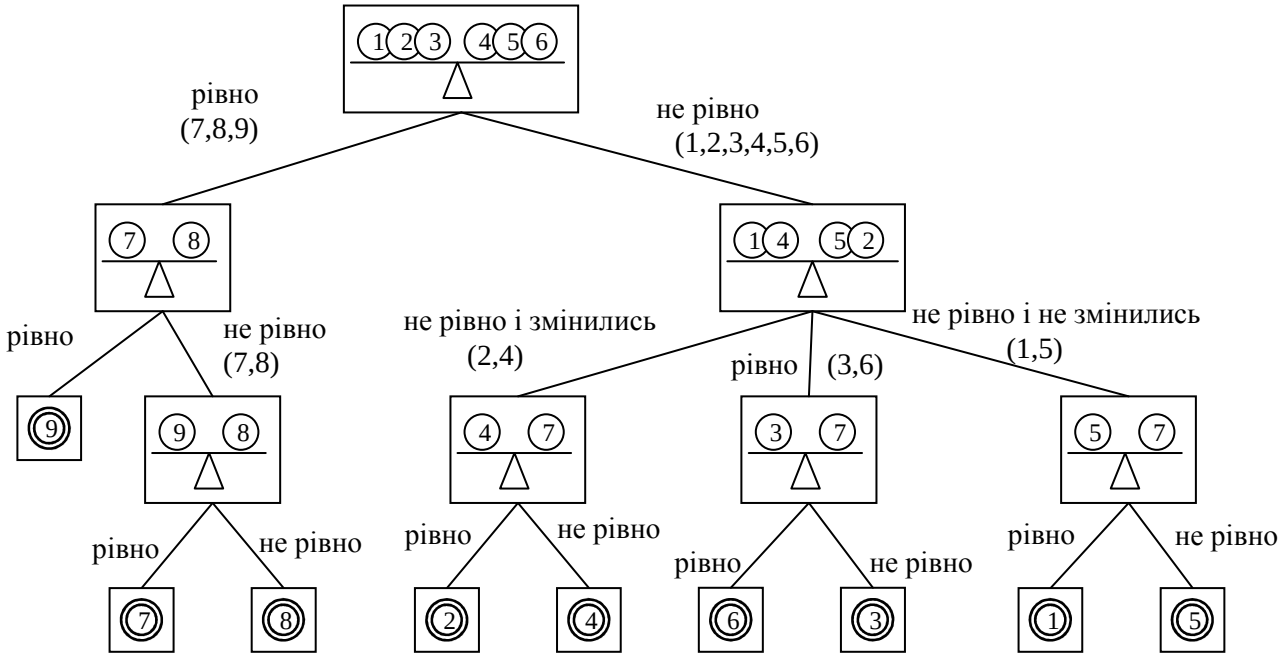

Рис. 28.7.

Розв'язок подано на рис. 28.7. Занумеруємо монети числами від 1 до 9. На першому кроці зважуємо групи монет 1, 2, 3 та 4, 5, 6. Якщо виявиться, що ці групи рівні за вагою, то тоді фальшива монета знаходиться серед монет 7, 8 9 (у дереві в дужках біля дуг зазначені ті монет, серед яких є фальшива) і ми переходимо до відповідної вершини дерева рішень. Тут ми зважуємо між собою монети 7 та 8. Якщо їх вага рівна, то тоді фальшива монета – 9. В іншому випадку фальшива монета – або 7, або 8, тож на третьому кроці зважуємо одну з них (8) з монетою, про яку точно відомо, що вона не є фальшивою (9). Якщо вони рівні, то тоді фальшива – 7, інакше – 8. Розглядаючи другий варіант при першому зважуванні, отримаємо, що фальшива монета міститься серед 1, 2, 3, 4, 5, 6. На наступному кроці ми знімаємо з кожної шальки по одній монеті, а ще по одній монеті перекладаємо на протилежну шальку. Тепер можливі три варіанти зважувань: рівно (фальшива монета – або 3, або 6), не рівно і стрілка терезів не змінилась (фальшива монета серед тих, які залишились на своїх місцях – або 1, або 5), не рівно і стрілка змінилась (фальшива монета серед тих, які перекладались на протилежні шальки – або 2, або 4). Аналогічно продовжуємо далі.

Зазначимо, що також за три кроки можна знайти фальшиву монету серед 13 (а якщо монет 12, то – додатково визначити, чи є вона важча, чи ні). Пропонуємо читачеві розв'язати ці задачі самостійно аналогічним способом.

У багатьох задачах потрібно приймати рішення стосовно досліджуваних об'єктів, відносячи їх до певних класів, тобто надаючи цим об'єктам класифікаційних ознак. Якщо ці ознаки наперед відомі, такі задачі називають **задачами класифікації**. Для об'єктів, щодо яких рішення вже прийнято, узагалі кажучи, невідомі правила отримання ними класифікаційних ознак. Тому задача прийняття рішень відносно нових об'єктів полягає в пошуку правил, за якими надано такі ознаки.

Розглянемо таблицю даних, рядки якої – це характеристики певного об'єкту чи явища. Нехай a – ім'я стовпця таблиці, яке називають **умовним атрибутом**; множину всіх умовних атрибутів цієї таблиці позначають як A. У стовпці містяться значення відповідного умовного атрибуту; множину цих значень позначають як E. Таку таблицю називають **інформаційною системою**. Формально це пара $D = (W, A)$, де W – не порожня скінченна множина об'єктів або

явищ, A – множина умовних атрибутів. Кортеж значень умовних атрибутів кожного об'єкте з множини W називають **прикладом**.

Система прийняття рішень – це інформаційна система $D = (W, A \cup \{d\})$, де $d \notin A$ – **атрибут прийняття рішень**. Він може набувати декількох значень, однак найчастіше використовують двійкові значення $\{0, 1\}$. Систему прийняття рішень можна подати у вигляді таблиці, у якій останній стовпець позначено атрибутом прийняття рішень. Її називають **таблицею прийняття рішень**.

Розглянемо задачу класифікації, як задачу прийняття рішень. У такій постановці задача побудови дерева рішень має наступний вигляд: значення атрибуту прийняття рішень d задає певний клас, до якого належить об'єкт із множини W, а множина значень атрибуту d розбиває множину W на класи, кожний з яких задає прийняте рішення. Таблиці прийняття рішень поставимо у відповідність дерево прийняття рішень, внутрішнім вершинам якого відповідають перевірки значень атрибутів із множини умовних атрибутів A. Ребрам приписано значення атрибутів, що містяться в цих вершинах, а листки такого дерева відповідають класам.

Нині для побудови дерев рішень на основі таблиць прийняття рішень активно застосовують алгоритм ID3, запропонований 1983 р. Куїнланом. Цей алгоритм будує дерево рішень і на основі прикладів генерує правила.

Алгоритм ID3 будує дерево рішень від кореня. Кожній внутрішній вершині (включно з коренем) ставлять у відповідність певну перевірку з множини перевірок. Для побудови перевірок використовують той факт, що значення кожного умовного атрибуту дає змогу розбити множину прикладів на підмножини, у яких усі приклади мають однакове значення цього атрибут. Якщо рекурсивно застосовувати це під час побудови дерева, ставлячи у відповідність кожній внутрішній вершині дерева підмножину, отриману внаслідок розбиття, то для кожної з одержаних підмножин буде побудовано нове піддерево. Процес побудови піддерев зупиняється, коли кожна підмножина складається лише з елементів з однаковими значеннями атрибута прийняття рішень. Листок дерева рішень задає клас, у який потрапляє об'єкт із множини W.

Наведемо кроки застосування алгоритму ID3 під час побудови дерева рішень згори донизу від кореня до листків. Ці кроки застосовують рекурсивно в кореневій і в кожній із внутрішніх вершин дерева прийняття рішень. Процес продовжують доти, доки не буде реалізовано крок 1 для всіх вершин дерева. Множину всіх умовних атрибутів позначено A.

Алгоритм ID3

1. Якщо вершині дерева прийняття рішень поставлено у відповідність множину прикладів R і всі ці приклади мають однакове значення атрибута прийняття рішень d, то цю вершину визначають як листок дерева та позначають значенням d.
2. Якщо для певної вершини умови кроку 1 не виконано, то розглядають множину умовних атрибутів A. Якщо $A \neq \varnothing$ то вибирають довільний атрибут $a \in A$; нехай множна його значень $E_a = \{e_1, \ldots, e_k\}$. Цю вершину позначають атрибутом a, сам атрибут a вилучають із множини A та виконують такі дії:
   - у вершині a утворюють ребра $e_1, \ldots, e_k$ (їх кількість становить $|E_a|$, і кожне ребро позначають елементом множини E_a);
   - множину прикладів R вершини a розбивають на підмножини з однаковим значенням атрибуту a (усього таких підмножин k, значення атрибуту a в першій підмножині дорівнює e_1, у другій – e_2, …, у k-й – e_k);
   - кінцю ребра e_j ставлять у відповідність підмножину прикладів, у яких значення атрибуту a дорівнює e_j.
3. Якщо на кроці 2 виявиться, що $A = \varnothing$, то щодо вершини, яка має стати листком, приймають спеціальне рішення залежно від специфіки задачі.

Отже, кожна внутрішня вершина являє собою корінь піддерева, якому відповідають усі приклади з однаковим значенням одного з атрибутів і різними значеннями атрибуту прийняття рішень. Кожному листку дерева відповідають приклади, які мають однакові значення одного з атрибутів і однакові значення атрибуту прийняття рішень.

Побудуємо дерево рішень для наведеної в табл. 28.1 інформації щодо проведення змагань з тенісу залежно від погоди.

День	**Погода**	**Температура**	**Вологість**	**Вітер**	**Гра**
Д1	Сонце	Спека	Висока	Слабкий	Ні
Д2	Сонце	Спека	Висока	Сильний	Ні
Д3	Хмари	Спека	Висока	Слабкий	Так
Д4	Дощ	Помірно	Висока	Слабкий	Так
Д5	Дощ	Холод	Норма	Слабкий	Так
Д6	Дощ	Холод	Норма	Сильний	Ні
Д7	Хмари	Холод	Норма	Сильний	Так
Д8	Сонце	Помірно	Висока	Слабкий	Ні
Д9	Сонце	Холод	Норма	Слабкий	Так
Д10	Дощ	Помірно	Норма	Слабкий	Так
Д11	Сонце	Помірно	Норма	Сильний	Так
Д12	Хмари	Помірно	Висока	Сильний	Так
Д13	Хмари	Спека	Норма	Слабкий	Так

Табл. 28.1.

Гра – це атрибут прийняття рішень. Множина всіх умовних атрибутів A = {«погода», «температура», «вологість», «вітер»} відповідає кореневій вершині. Виберемо атрибут «погода» та позначимо ним кореневу вершину. Множина значень цього атрибуту складається з трьох елементів: сонце, хмари, дощ. Кореневій вершині поставимо у відповідність три ребра, кожному з яких припишемо значення атрибуту «погода». Множину прикладів розіб'ємо на три підмножини, які відповідають значенням атрибуту «погода»; ці підмножини відповідають кожній із вершин 2, 3, 4 дерева, зображеного на рис. 28.8. Атрибут «погода» вилучимо з множини A та отримаємо множину A = {«температура», «вологість», «вітер»}.

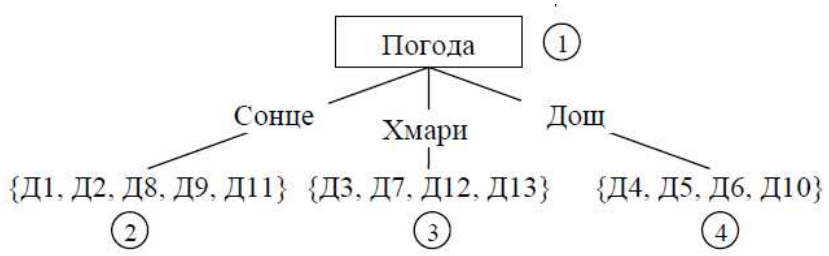

Рис. 28.8. Перший крок роботи алгоритму ID3.

Розглянемо вершину з номером 2. Їй відповідає підмножина прикладів {Д9, Д11}, які мають значення атрибуту прийняття рішення «так», і підмножина прикладів {Д1, Д2, Д8}, які мають значення атрибуту прийняття рішення «ні». Виберемо наступний атрибут із множини A; нехай це «температура». Позначимо ним вершину 2, побудуємо три ребра зі значеннями цього атрибуту та розіб'ємо множину прикладів у вершині 2 на три підмножини, у кожній з яких значення температури однакові.

На рис. 28.9 у вершині 5 приклади Д1 та Д2 мають однакові значення атрибуту «гра» – «ні». Тому цю вершину позначимо «ні», і вона стане листком. Аналогічно, вершину 7 позначимо «так», і вона також стане листком. Вилучимо атрибут «температура» з множини A. Одержимо множину A = {«вологість», «вітер»}. Вершину 6 позначимо атрибутом «вологість».

На рис. 28.10 показано два листки, які відповідають значенням «так» і «ні» відповідно для прикладів Д11 і Д8. Вилучимо атрибут «вологість» із множини *A* й отримуємо *A* = {«вітер»}.

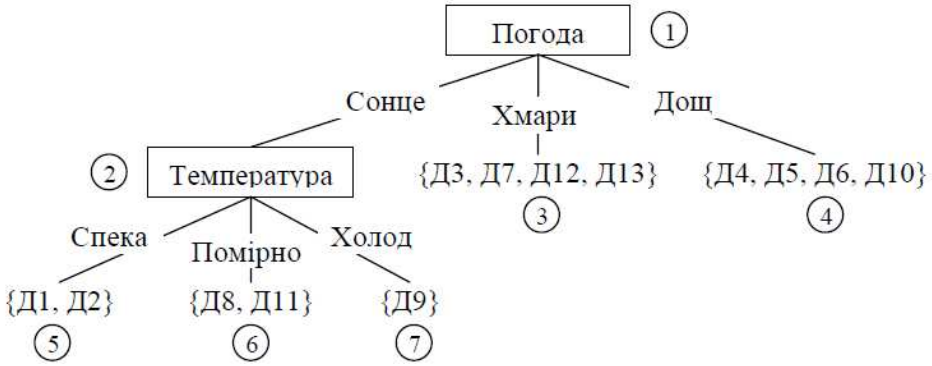

Рис. 28.9. Вилучення атрибуту «температура».

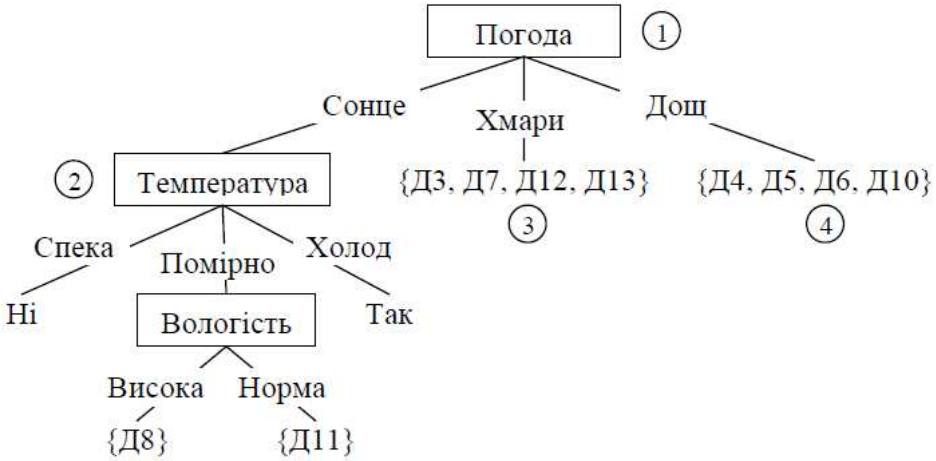

Рис. 28.10. Вилучення атрибуту «вологість».

Тепер розглянемо вершину 3 на рис. 28.8. Усі приклади, приписані цій вершині, мають значення «так» атрибуту «гра», тому ця вершина є листком; позначимо її «так». У вершині 4 приклади мають різні значення атрибуту «гра»; множина *A* = {«температура», «вологість», «вітер»}. Виберемо атрибут «вітер», який має два значення: «слабкий» і «сильний». Ці значення припишемо ребрам, інцидентним вершині 4. Множину прикладів розіб'ємо на дві підмножини: {Д4, Д5, Д10} зі значенням «так» і {Д6} зі значенням «ні». Остаточне дерево рішень набуде вигляду, зображеного на рис. 28.11.

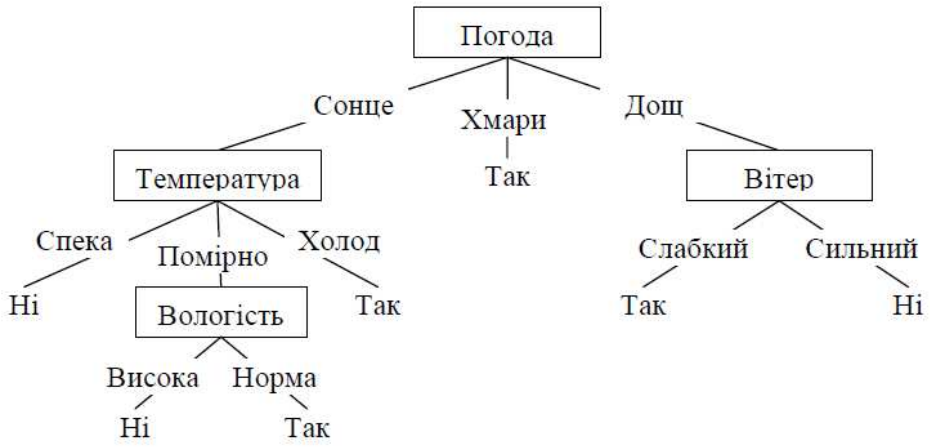

Рис. 28.11. Дерево рішень для таблиці 28.1.

За побудованим деревом прийняття рішень можемо визначити множину правил:

1) якщо (погода=сонце) і (температура=спека), то (гра=ні);
2) якщо (погода=сонце) і (температура=помірно) і (вологість=висока), то (гра=ні);
3) якщо (погода=сонце) і (температура=помірно) і (вологість=норма), то (гра=ні);
4) якщо (погода=сонце) і (температура=холод), то (гра=так);
5) якщо (погода=хмари), то (гра=так);
6) якщо (погода=дощ) і (вітер=слабкий), то (гра=так);
7) якщо (погода=дощ) і (вітер=сильний), то (гра=ні).

Зрозуміло, що дерево прийняття рішень буде набувати різних виглядів в залежності від послідовності обирання атрибутів на кожній ітерації алгоритму ID3. Зазвичай атрибути впорядковуються за важливістю, яку може заздалегідь визначити експерт в галузі проблеми задачі.

Алгоритм ID3 – один із найважливіших методів індуктивного відновлення правил за прикладами, який забезпечує автоматичну побудову баз знань діагностичних експертних систем.