Розділ 4

Дерева та їх застосування

- Основні означення та властивості
- Рекурсія. Обхід дерев.
  Префіксна та постфіксна форми запису виразів
- Бінарне дерево пошуку
- Дерево прийняття рішень
- Бектрекінг (пошук із поверненнями)
- Каркаси (з'єднувальні дерева)

Поняття дерева широко застосовують у багатьох розділах математики й інформатики. Наприклад, дерева використовують як інструмент обчислень, зручний спосіб збереження даних, їх сортування чи пошуку.

4.1. Основні означення та властивості

Деревом називають зв'язний граф без простих циклів. Граф, який не містить простих циклів і складається з k компонент, називають *лісом* із k дерев.

Приклад 4.1. На рис. 4.1 зображено приклади дерев. Граф, зображений на рис. 4.2 — не дерево, бо він незв'язний.

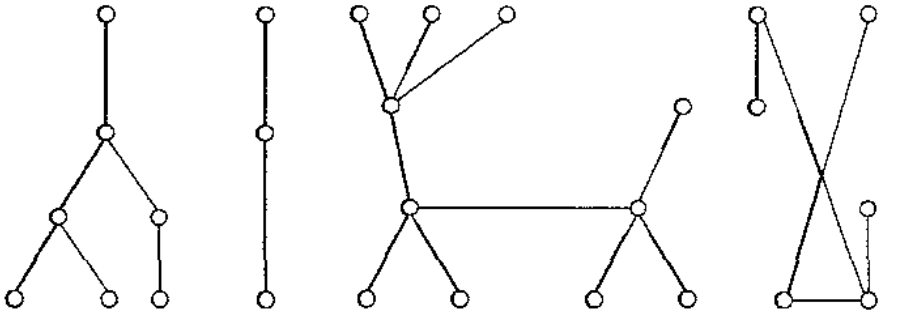


Рис. 4.1

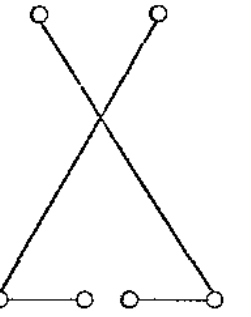


Рис. 4.2

Зауваження. З означення випливає, що дерева й ліси являють собою прості графи.

ТЕОРЕМА 4.1. Нехай граф T має n вершин. Тоді такі твердження еквівалентні:

(I) граф T – дерево;

(II) граф T не містить простих циклів і має $(n-1)$ ребро;

(III) граф T зв'язний і має $(n-1)$ ребро;

(IV) граф T зв'язний, але вилучення довільного ребра робить його незв'язним;

(V) довільні дві вершини графа T з'єднані точно одним простим шляхом;

(VI) граф T не містить простих циклів, але, додавши до нього довільне нове ребро, ми отримаємо точно один простий цикл.

Доведення (математичною індукцією). У разі $n=1$ твердження тривіальні; припустимо, що $n \geqslant 2$.

(I) → (II). За означенням T не містить простих циклів. Отже, вилучивши довільне ребро, ми одержимо два графи, кожний з яких являє собою дерево з меншою, ніж у T, кількістю вершин. За припущенням індукції кількість ребер у кожному з отриманих дерев на 1 менша за кількість вершин. Звідси випливає, що граф T має $(n-1)$ ребро.

(II) → (III). Припустимо, що граф T незв'язний. Тоді кожна його компонента являє собою зв'язний граф без простих циклів, тобто дерево. Звідси випливає, що кількість вершин у кожній компоненті на одиницю більша від кількості ребер. Отже, загальна кількість вершин графа T більша за кількість ребер принаймні на 2. Але це суперечить тому, що граф T має $(n-1)$ ребро.

(III) → (IV). Вилучимо довільне ребро, отримаємо граф з n вершинами та $(n-2)$ ребрами. Припущення про зв'язність такого графа суперечить теоремі про оцінку (знизу) кількості ребер звичайного графа (теорема 3.6).

(IV) → (V). Оскільки граф T зв'язний, то кожну пару його вершин з'єднано принаймні одним простим шляхом (теорема 3.4). Якщо якусь пару вершин з'єднано двома простими шляхами, вони замикаються в простий цикл. Але це суперечить тому, що вилучення довільного ребра робить граф T незв'язним.

(V) → (VI). Припустимо, що граф T містить простий цикл. Тоді довільні дві вершини цього циклу з'єднано принаймні двома простими шляхами, що суперечить твердженню (V). Додавши тепер до графа T ребро e, одержимо єдиний простий цикл, бо інцидентні ребру e вершини вже з'єднано в графі T точно одним простим шляхом.

(VI) → (I). Припустимо, що граф T незв'язний. Тоді додавання будь-якого ребра, що з'єднує вершину однієї компоненти з вершиною іншої, не зумовлює утворення простого циклу, що суперечить твердженню (VI).

Наслідок із твердження (II). Ліс із k дерев, який містить n вершин, має $(n-k)$ ребер.

У багатьох застосуваннях певну вершину дерева означають як *корінь*. Тоді можна природно приписати напрямок кожному ребру. Оскільки існує єдиний простий шлях від кореня до кожної вершини графа, то можна орієнтувати кожне ребро в напрямку від кореня. Отже, дерево разом із виділеним коренем утворює орієнтований граф, які називають *кореневим деревом*.

Різні способи вибору кореня дають змогу утворити різні кореневі дерева.

Приклад 4.2. На рис. 4.3, *а* зображено дерево, а на рис. 4.3, *б*, *в* — кореневі дерева з коренями відповідно у вершинах *a* та *c*.

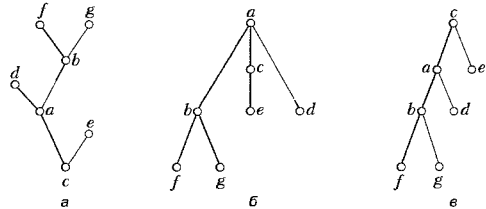


Рис. 4.3

Нехай T — кореневе дерево. Якщо v — його вершина, відмінна від кореня, то її *батьком* називають єдину вершину u таку, що є орієнтоване ребро (u, v). Якщо u — батько, то v — *син*. Аналогічно за генеалогічною термінологією можна означити інших предків і нащадків вершини v. Вершини дерева, які не мають синів, називають *листками*. Вершини, які мають синів, називають *внутрішніми*. Нехай a — вершина дерева. Тоді *піддеревом* із коренем a називають підграф, що містить a та всі вершини — нащадки вершини a, а також інцидентні їм ребра.

Кореневе дерево називають *m-арним*, якщо кожна його внутрішня вершина має не більше ніж m синів. Дерево називають *повним m-арним*, якщо кожна його внутрішня вершина має точно m синів. У разі $m=2$ дерево називають *бінарним*.

Кореневе дерево, у якому сини кожної внутрішньої вершини впорядковано, називають *упорядкованим*. Таке дерево зображають так, щоб сини кожної вершини були розміщені зліва направо.

Якщо внутрішня вершина впорядкованого бінарного дерева має двох синів, то першого називають *лівим*, а другого — *правим*. Піддерево з коренем у вершині, яка являє собою лівого сина вершини v, називають *лівим піддеревом у цій вершині*. Якщо корінь піддерева — правий син вершини v, то таке піддерево називають *правим піддеревом у цій вершині*.

Приклад 4.3. У дереві, зображеному на рис. 4.4, Л і П — відповідно ліве та праве піддерева у вершині *c*.

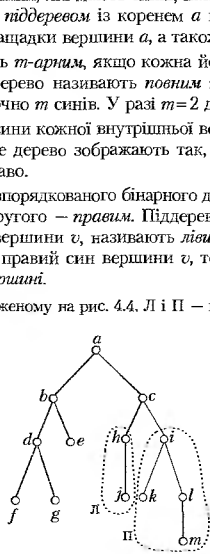


Рис. 4.4

ТЕОРЕМА 4.2. Повне m-арне дерево з i внутрішніми вершинами містить $n=mi+1$ вершин.

Доведення. Кожна вершина, окрім кореня, — син внутрішньої вершини. Оскільки кожна з i внутрішніх вершин має m синів, то всього є, якщо не враховувати корінь, mi вершин, а з урахуванням кореня їх $mi+1$.

Рівнем вершини v в кореневому дереві називають довжину простого шляху від кореня до цієї вершини (цей шлях, очевидно, єдиний). Рівень кореня вважають нульовим. *Висотою* кореневого дерева називають максимальний із рівнів його вершин. Інакше кажучи, висота кореневого дерева — це довжина найдовшого простого шляху від кореня до будь-якої вершини. Повне m-арне дерево, у якого всі листки на одному рівні, називають *завершеним*.

Кореневе m-арне дерево з висотою h називають *збалансованим* [52], якщо всі його листки на рівнях h або $h-1$.

Приклад 4.4. На рис. 4.5 зображено збалансоване бінарне дерево, яке має висоту 4: усі його листки на рівнях 3 та 4.

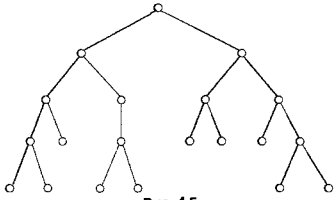


Рис. 4.5

Збалансованість можна означити й інакше [7, 35].

ТЕОРЕМА 4.3. Нехай m-арне дерево має висоту h. Тоді в ньому не більше ніж m^h листків.

Доведення. Застосуємо математичну індукцію за h. У разі $h=1$ твердження очевидне. Припустимо, що воно справджується для всіх m-арних дерев із меншою висотою, ніж h. Нехай T — m-арне дерево з висотою h. Тоді його листки — це листки піддерев, отриманих із T вилученням ребер, що з'єднують корінь дерева T з кожною вершиною рівня 1 (рис. 4.6).

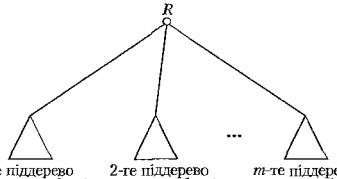


Рис. 4.5

Кожне з цих піддерев має не більшу висоту, ніж $h-1$. За індуктивною гіпотезою всі вони мають не більше ніж m^{h-1} листків. Позаяк таких піддерев не більше ніж m, то загальна кількість листків у дереві T не перевищує $mm^{h-1}=m^h$.

Наслідок. Якщо m-арне дерево з висотою h має l листків, то $h \geqslant \lceil \log_m l \rceil$. Якщо m-арне дерево повне та збалансоване, то $h=\lceil \log_m l \rceil$. Нагадаємо, що $\lceil x \rceil$ — це найменше ціле число, яке більше чи дорівнює x.

Доведення. За теоремою 4.3 $l \leqslant m^h$. Прологарифмуємо цю нерівність за основою m: $\log_m l \leqslant h$. Оскільки h — ціле, то $h \geqslant \lceil \log_m l \rceil$. Тепер припустимо, що дерево повне та збалансоване. Вилучимо всі листки на рівні h (разом з інцидентними їм ребрами). Одержимо завершене m-арне дерево висотою $h-1$. Воно має m^{h-1} листків (див. задачу 12). Отже, $m^{h-1} < l \leqslant m^h$. Звідси випливає, що $h-1 < \log_m l \leqslant h$, тобто $h=\lceil \log_m l \rceil$.

4.2. Рекурсія. Обхід дерев. Префіксна та постфіксна форми запису виразів

Об'єкт називають *рекурсивним*, якщо він містить сам себе чи його означено за допомогою самого себе. Рекурсія — потужний засіб у математичних означеннях.

Приклад 4.5. У підрозділі 4.1 сформульовано означення повного бінарного дерева. Тепер означимо його рекурсивно:

а) o (ізольована вершина) — *повне бінарне дерево*;

б) якщо A та B — повні бінарні дерева, то конструкція, зображена на рис. 4.7, — *повне бінарне дерево*.

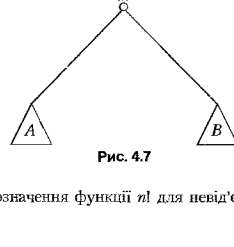


Рис. 4.7

Приклад 4.6. Рекурсивне означення функції $n!$ для невід'ємних цілих чисел має такий вигляд:

а) $0!=1$;

б) якщо $n>0$, то $n!=n(n-1)!$.

Очевидно, що важливість рекурсії пов'язана з тим, що вона дає змогу означити нескінченну множину об'єктів за допомогою скінченного висловлювання. Так само нескінченні обчислення можна описати за допомогою скінченної рекурсивної програми, навіть якщо вона не містить явних циклів. Проте найдоцільніше ви-

користовувати рекурсивні алгоритми тоді, коли розв'язувану задачу, обчислювану функцію чи оброблювані дані задано за допомогою рекурсії.

Чимало задач можна моделювати з використанням кореневих дерев. Поширене таке загальне формулювання задачі: виконати задану операцію Φ з кожною вершиною дерева. Тут Φ – параметр загальнішої задачі відвідування всіх вершин, або так званого *обходу дерева*. Розглядаючи розв'язування цієї задачі як єдиний послідовний процес відвідування вершин дерева в певному порядку, можна вважати їх розміщеними одна за одною. Опис багатьох алгоритмів істотно спрощується, якщо можна говорити про наступну вершину дерева, маючи на увазі якесь упорядкування. Є три принципи впорядкування вершин, які природно випливають зі структури дерева. Як і саму деревоподібну структуру, їх зручно формулювати за допомогою рекурсії.

Звертаючись до бінарного дерева, де R – корінь, A та B – ліве та праве піддерева (рис. 4.7), можна означити такі впорядкування.

1. Обхід *у прямому порядку* (*preorder*), або *зверху вниз*: R, A, B (корінь відвідують до обходу піддерев).
2. Обхід *у внутрішньому порядку* (*inorder*), або *зліва направо*: A, R, B.
3. Обхід *у зворотному порядку* (*postorder*), або *знизу вверх*: A, B, R (корінь відвідують після обходу піддерев).

Приклад 4.7. На рис. 4.8 зображено бінарне дерево. Різні обходи дадуть такі послідовності вершин:

- обхід у прямому порядку: *a b d e h o c f m p q*;
- обхід у внутрішньому порядку: *d b h e o a f c p m q*;
- обхід у зворотному порядку: *d h o e b f p q m c a*.

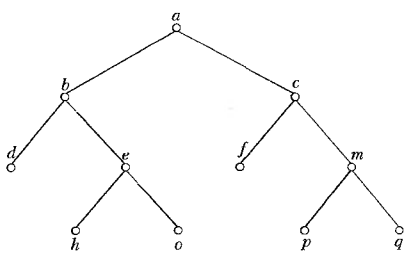


Рис. 4.8

Зазначені способи обходу бінарних дерев можна узагальнити й на довільні m-арні дерева. Обхід таких дерев у прямому порядку (зверху вниз) схематично зображено на рис. 4.9, у внутрішньому порядку (зліва направо) – на рис. 4.10, у зворотному (знизу вверх) – на рис. 4.11.

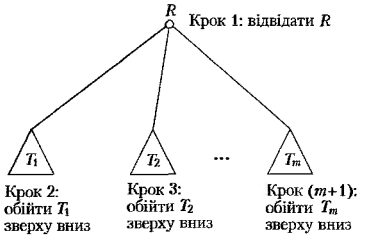


Рис. 4.9

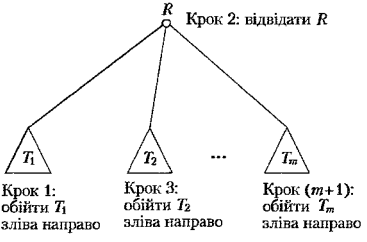


Рис. 4.10

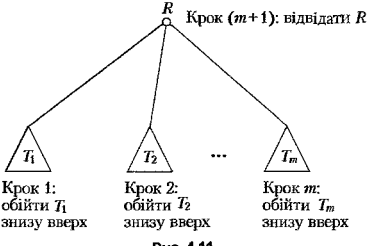


Рис. 4.11

Надзвичайно поширене в інформатиці застосування обходу дерев — зіставлення виразам (арифметичним, логічним тощо) дерев і побудова на цій основі різних форм запису виразів. Суть справи зручно пояснити на прикладі. Розглянемо арифметичний вираз

$$\left(a+\frac{b}{c}\right)*(d-e*f).$$

Подамо його у вигляді дерева. Послідовність дій відтворено на рис. 4.12. Рамкою на ньому обведено дерево, яке відповідає заданому арифметичному виразу. Внутрішнім вершинам цього дерева відповідають символи операцій, а листкам — операнди.

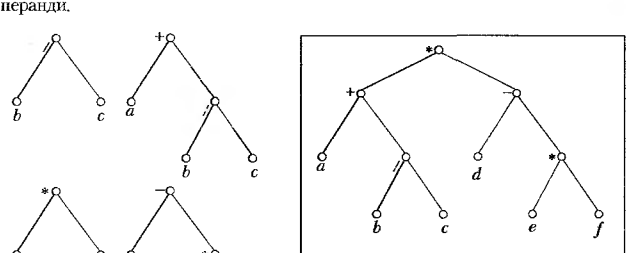


Рис. 4.12

Обійдемо це дерево, записуючи символи у вершинах у тому порядку, у якому вони зустрічаються в разі заданого способу обходу. Отримаємо такі три послідовності:

- у разі обходу в прямому порядку — *префіксний* (*польський*) *запис*

$$*+a\,/\,b\,c-d*e\,f;$$

- у разі обходу у внутрішньому порядку — *інфіксний запис* (поки що без дужок, потрібних для визначення порядку операцій)

$$a+b\,/\,c*d-e*f;$$

- у разі обходу в зворотному порядку — *постфіксний* (*зворотний польський*) *запис*

$$a\,b\,c\,/+d\,e\,f*-*.$$

Звернімося спочатку до інфіксної форми запису виразу. Без дужок вона неоднозначна: один запис може відповідати різним деревам. Наприклад, дереву, зображеному на рис. 4.13, у разі обходу зліва направо відповідає той самий вираз $a+b\,/\,c*d-e*f$, що й дереву на рис. 4.12 (у рамці), хоча на цих рисунках зображено різні дерева. Щоб уникнути неоднозначності інфіксної форми, використовують круглі дужки щоразу, коли зустрічають операцію. Повністю „одужкований" вираз, одержаний під час обходу дерева у внутрішньому порядку, називають *інфіксною формою* запису. Отже, для дерева з рис. 4.12 інфіксна форма така: $((a+(b\,/\,c))*(d-(e*f)))$; для дерева, зображеного на рис. 4.13, інфіксна форма має такий вигляд: $(a+(((b\,/\,(c*d))-e)*f))$.

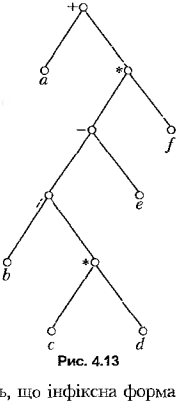


Рис. 4.13

Наведені міркування свідчать, що інфіксна форма запису виразів незручна. На практиці використовують префіксну та постфіксну форми, бо вони однозначно відповідають виразу й не потребують дужок. Ці форми запису називають *польським записом* (на честь польського математика й логіка Яна Лукасевича, українця за походженням).

Приклад 4.8. Розглянемо логічний вираз $(\neg(p \wedge q)) \sim (\neg p \vee \neg q)$. Послідовні етапи побудови відповідного бінарного дерева зображено на рис. 4.14.

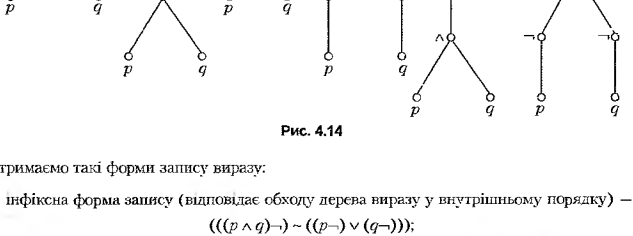


Рис. 4.14

Отримаємо такі форми запису виразу:

- інфіксна форма запису (відповідає обходу дерева виразу у внутрішньому порядку) — $(((p \wedge q)\neg) \sim ((p\neg) \vee (q\neg)))$;
- польський запис (відповідає обходу дерева виразу в прямому порядку) — $\sim \neg \wedge pq \vee \neg p \neg q$;
- зворотний польський запис (відповідає обходу дерева виразу в зворотному порядку) — $pq \wedge \neg p \neg q \neg \vee \sim$.

Для обчислення значення виразу в польському записі його проглядають справа наліво та знаходять два операнди разом зі знаком операції перед ними. Ці операнди та знак операції вилучають із запису, виконують операцію, а її результат записують на місце вилучених символів.

Приклад 4.9. Обчислимо значення виразу в польському записі (стрілка означає піднесення до степеня)

$$+-*2\,3\,5\,/\uparrow 2\,3\,4.$$

За сформульованим правилом виділимо $\uparrow 2\,3$, ці символи вилучимо й обчислимо $2\uparrow 3=8$; результат запишемо на місце вилучених символів:

$$+-*2\,3\,5\,/\,8\,4.$$

Продовжимо обчислення. Динаміку процесу відображено в табл. 4.1.

Таблиця 4.1

Крок	Вираз	Виділені символи	Виконання операції
1	+−*235/↑234	↑23	2↑3=8
2	+−*235/84	/84	8/4=2
3	+−*2352	*23	2*3=6
4	+−652	−65	6−5=1
5	+12	+12	1+2=3
6	3		

Для обчислення значення виразу в зворотному польському записі його проглядають зліва направо та виділяють два операнди разом зі знаком операції після них. Ці операнди та знак операції вилучають із запису, виконують операцію, а її результат записують на місце вилучених символів.

Приклад 4.10. Обчислимо значення виразу в зворотному польському записі

$$7\,2\,3*-4\uparrow 9\,3\,/+.$$

Динаміку обчислень відображено в табл. 4.2.

Таблиця 4.2

Крок	Вираз	Виділені символи	Виконання операції
1	723*−4↑93/+	23*	2*3=6
2	76−4↑93/+	76−	7−6=1
3	14↑93/+	14↑	1↑4=1
4	193/+	93/	9/3=3
5	13+	13+	1+3=4
6	4		

Оскільки польські записи однозначні та їх значення можна легко обчислити без сканування назад і вперед, їх широко використовують у комп'ютерних науках, особливо для конструювання компіляторів.

4.3. Бінарне дерево пошуку

Бінарне дерево забезпечує 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. підрозділ 5.3).

Під час побудови бінарного дерева пошуку використовують його рекурсивну властивість, яку можна описати так. 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 (рис. 4.15, 4.16).

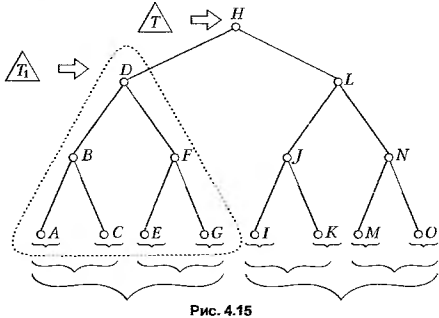


Рис. 4.15

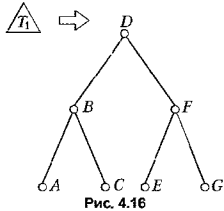


Рис. 4.16

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

Наведемо кроки алгоритму.

Крок 1. Почати з кореня.

Крок 2. Якщо об'єкт менший, ніж ключ у вершині, то перейти до лівого сина.

Крок 3. Якщо об'єкт більший, ніж ключ у вершині, то перейти до правого сина.

Крок 4. Повторювати кроки 2 та 3, доки не досягнемо вершини, яку не визначено (тобто її немає).

Крок 5. Якщо досягнуто невизначену вершину, то визначити (тобто додати) вершину з новим об'єктом як ключем.

Приклад 4.11. Побудуємо бінарне дерево пошуку для такого списку слів в українському алфавіті: математика, фізика, географія, зоологія, метеорологія, біологія, психологія, хімія. Процес побудови зображено на рис. 4.17.

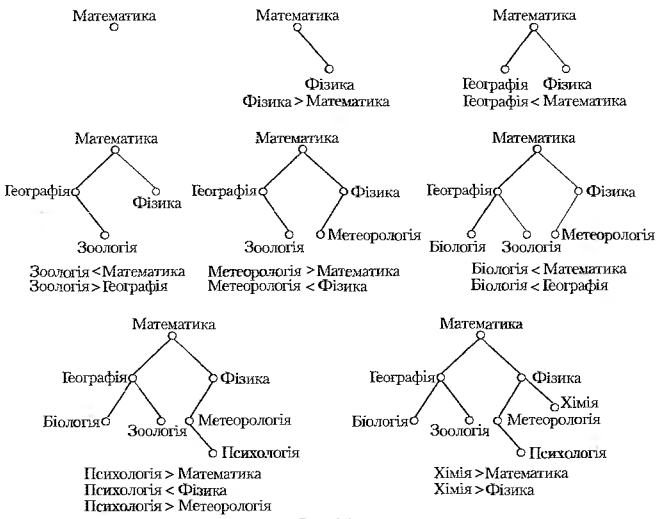


Рис. 4.17

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'єкта в бінарному дереві.

Алгоритм пошуку об'єкта в дереві

Наведемо кроки алгоритму.

Крок 1. Почати з кореня.

Крок 2. Якщо об'єкт менший, ніж ключ у вершині, то перейти до лівого сина.

Крок 3. Якщо об'єкт більший, ніж ключ у вершині, то перейти до правого сина.

Крок 4. Якщо об'єкт дорівнює ключу у вершині, то об'єкт знайдено; виконати потрібні дії й вийти.

Крок 5. Повторювати кроки 2, 3 та 4, доки не досягнемо вершини, яку не визначено.

Крок 6. Якщо досягнуто невизначену вершину, то даний об'єкт не зберігається в дереві; виконати потрібні дії й вийти.

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. 4.18, де вершини без ключів позначено подвійними кружечками.

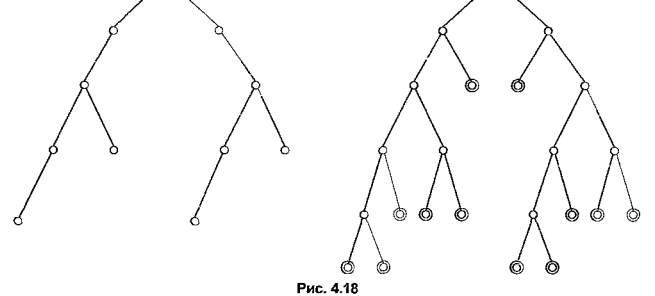


Рис. 4.18

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'єкта, дорівнює довжині найдовшого шляху в U від кореня до листка, тобто висоті дерева. Внутрішні вершини дерева U – це (всі) вершини дерева T. Отже, дерево U має n внутрішніх вершин. За теоремою 4.2 кількість листків у ньому дорівнює $2n+1-n=n+1$. Згідно з наслідком із теореми 4.3 висота дерева U більша чи дорівнює $\lceil \log(n+1) \rceil$. Отже, потрібно щонайменше $\lceil \log(n+1) \rceil$ порівнянь, щоб додати чи локалізувати довільний об'єкт. Якщо дерево збалансоване, то його висота дорівнює $\lceil \log(n+1) \rceil$. Отже, у цьому разі для додавання чи локалізації об'єкта потрібно не більше ніж $\lceil \log(n+1) \rceil$ порівнянь.

Бінарне дерево пошуку може розбалансуватись унаслідок додавання нових об'єктів, тому потрібен алгоритм *ребалансування* (тобто відновлення збалансованості). Проте процедура додавання об'єкта, 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. Такс означення збалансованості широко використовують на практиці; його ввели 1962 р. Г. М. Адельсон-Вельський і Є. М. Ландіс. Приклад АВЛ-дерева наведено на рис. 4.19.

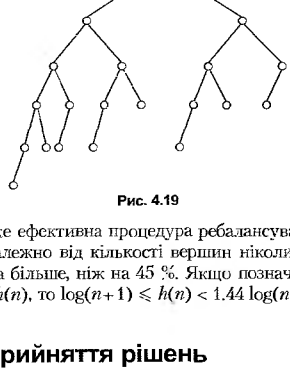


Рис. 4.19

Для АВЛ-дерев є дуже ефективна процедура ребалансування [7]. 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 $\log(n+1) \leqslant h(n) < 1.44 \log(n+2) - 0.328$.

4.4. Дерево прийняття рішень

Кореневі дерева можна застосовувати для розв'язування задач прийняття рішень. Для цього використовують *дерева прийняття рішень*, або *дерева рішень* [24, 29]. Кожному листку такого дерева поставлено у відповідність рішення зі скінченної множини відомих рішень, 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.

Приклад 4.12. Серед восьми монет одна фальшива, вона має меншу вагу. Знайдемо цю монету зважуваннями на балансових терезах за допомогою якнайменшої кількості зважувань. Розв'язок подано на рис. 4.20. Занумеруємо монети числами 1, 2, ..., 8. Порівнюємо вагу груп монет 1, 2, 3 та 4, 5, 6. Якщо одна із цих груп виявиться легшою, то це означає, що фальшива монета є в цій групі, і її можна виявити другим зважуванням. У разі балансу терез фальшива одна з монет 7, або 8, що також можна виявити другим зважуванням. Очевидно, що кількість зважувань дорівнює висоті дерева, тобто двом (рис. 4.21).

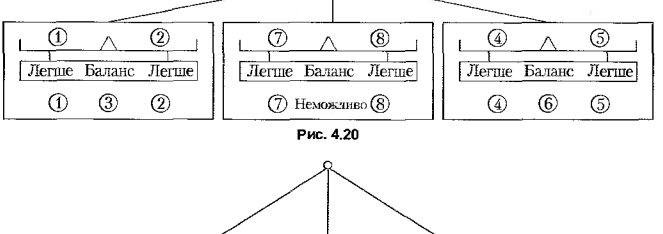


Рис. 4.20

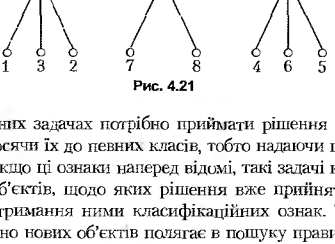


Рис. 4.21

У багатьох практичних 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 – характеристики певного об'єкта 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 атрибута; множину цих значень позначають як E_a. Таку таблицю називають *інформаційною системою*. Формально це пара $D=(W, A)$, де W — непорожня скінченна множина об'єктів або явищ, A — множина *умовних*

атрибутів. Кортеж значень умовних атрибутів кожного об'єкта 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} або {Yes, No}. 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 такий вигляд: значення атрибута прийняття рішень d задає певний клас, до якого належить об'єкт із множини W, а множина значень атрибута 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.

Дерево рішень ставить у відповідність прикладу номер класу. Для цього використовують множину перевірок значень атрибутів даного прикладу у вершинах дерева на шляху від кореня до листка. Якщо атрибут прийняття рішень має декілька значень, то таке дерево може набути вигляду, наведеного на рис. 4.22. Тут 1, 2 та 3 — класифікаційні ознаки, або значення атрибута прийняття рішень. Наприклад, перевірці атрибута a_2 відповідають три його значення: ліве значення атрибута a_2 надає прикладу ознаку 3, середнє потребує для прийняття рішення перевірки атрибута a_4, а праве надає прикладу ознаку 1.

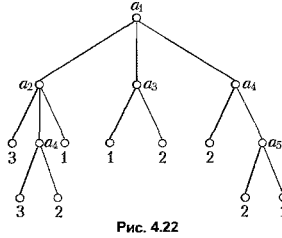


Рис. 4.22

Розглянемо приклад класифікації, виконаної за допомогою дерева прийняття рішень. На рис. 4.23 наведено класифікацію за атрибутами *температура* (зі значеннями *низька, висока, середня*) та *тиск* (зі значеннями *спадає, не змінюється* та *зростає*). Процедура прийняття рішення складається з побудови шляху від кореня до листків і перевірок, виконуваних у кожній внутрішній вершині. Ребра, які починаються в них вершинах, позначено можливими відповідями на питання перевірок. Результатами класифікації позначено листки дерева.

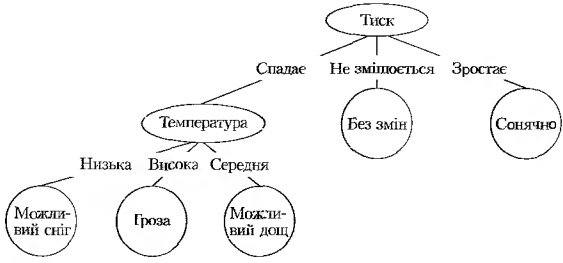


Рис. 4.23

Класифікатор записують у вигляді правил *if K then d=b*, де *K* — кон'юнкція підстановок вигляду *a=e* (під *підстановкою* тут розуміють надання атрибуту *a* значення *e*), *d=b* — результат класифікації (*d* — ім'я атрибута прийняття рішень, а *b* — його значення). Правила будують на шляху від кореня дерева до відповідного листка. Для дерева на рис. 4.23 множина правил така:

```
if (тиск=спадає) ∧ (температура=низька) then (погода=можливий сніг);
if (тиск=спадає) ∧ (температура=висока) then (погода=гроза);
if (тиск=спадає) ∧ (температура=середня) then (погода=можливий дощ);
if (тиск=не змінюється) then (погода=без змін);
if (тиск=зростає) then (погода=сонячно)
```

Нині для побудови дерев рішень на основі таблиць прийняття рішень активно застосовують алгоритм ID3, запропонований 1983 р. Куінланом (J. Quinlan). 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 атрибута дає змогу розбити множину прикладів на підмножини, у яких усі приклади мають однакове значення цього атрибута. Якщо рекурсивно застосовувати це під час побудови дерева, ставлячи у відповідність кожній внутрішній вершині дерева підмножину, отримаємо внаслідок розбиття, то для кожної з одержаних підмножин буде побудовано нове піддерево. Процес побудови піддерев зупиняють, 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 зверху вниз 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

Наведемо кроки алгоритму.

Крок 1. Якщо вершині дерева прийняття рішень поставлено у відповідність множину прикладів R і всі ці приклади мають однакове значення атрибута прийняття рішення d, то цю вершину визначають як листок дерева та позначають цим значенням d.

Крок 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 множина його значень $E_a=\{e_1, e_2, ..., e_k\}$. Цю вершину позначають атрибутом a, сам атрибут a вилучають із множини A й виконують такі дії:

- у вершині a утворюють ребра $e_1, e_2, ..., e_k$ (їх кількість становить $|E_a|$, і кожне ребро позначають елементом множини E_a);
- множину прикладів R вершини a розбивають на підмножини з однаковим значенням атрибута a (усього таких підмножин k, значення атрибута 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 атрибута a дорівнює e_j.

Крок 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 атрибута прийняття рішень. Кожному листку дерева відповідають приклади, що мають однакові значення одного з атрибутів і однакові значення атрибута прийняття рішень.

Приклад 4.13. Побудуємо дерево рішень для наведеної в табл. 4.3 інформації щодо проведення змагань із тенісу залежно від погоди. *Гра* — це атрибут прийняття рішень. Множина всіх умовних атрибутів $A=\{$*погода, температура, вологість, вітер*$\}$ відповідає кореневій вершині. Виберемо атрибут *погода* й позначимо ним кореневу вершину. Множина значень цього атрибута складається з трьох елементів: *сонце, хмари, дощ*. Кореневій вершині поставимо у відповідність три ребра, кожному з яких припишемо значення атрибута *погода*. Множину прикладів розіб'ємо на три підмножини, які відповідають значенням атрибута *погода*. Атрибут *погода* вилучимо з множини A й отримаємо множину $A=\{$*температура, вологість, вітер*$\}$, яка відповідає кожній із вершин 2, 3, 4 дерева, зображеного на рис. 4.24.

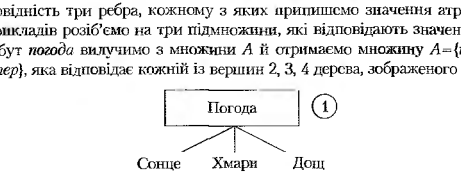


Рис. 4.24

Таблиця 4.3

ДЕНЬ	ПОГОДА	ТЕМПЕРАТУРА	ВОЛОГІСТЬ	ВІТЕР	ГРА
*D*1	Сонце	Спека	Висока	Слабкий	Ні
*D*2	Сонце	Спека	Висока	Сильний	Ні
*D*3	Хмари	Спека	Висока	Слабкий	Так
*D*4	Дощ	Помірно	Висока	Слабкий	Так
*D*5	Дощ	Холод	Норма	Слабкий	Так
*D*6	Дощ	Холод	Норма	Сильний	Ні
*D*7	Хмари	Холод	Норма	Сильний	Так
*D*8	Сонце	Помірно	Висока	Слабкий	Ні
*D*9	Сонце	Холод	Норма	Слабкий	Так
*D*10	Дощ	Помірно	Норма	Слабкий	Так
*D*11	Сонце	Помірно	Норма	Сильний	Так
*D*12	Хмари	Помірно	Висока	Сильний	Так
*D*13	Хмари	Спека	Норма	Слабкий	Так
*D*14	Дощ	Помірно	Висока	Сильний	Ні

Розглянемо вершину з номером 2. Їй відповідає підмножина прикладів {*D*9, *D*11}, які мають значення атрибута прийняття рішення *так*, і підмножина прикладів {*D*1, *D*2, *D*8}, які мають значення атрибута прийняття рішення *ні*. Виберемо наступний атрибут із множини *A*; нехай це *температура*. Позначимо ним вершину 2, побудуємо три ребра зі значеннями цього атрибута й розіб'ємо множину прикладів у вершині 2 на три підмножини, у кожній з яких значення температури однакові.

На рис. 4.25 у вершині 5 приклади *D*1 і *D*2 мають однакові значення (*ні*) атрибута *гра*. Тому цю вершину позначимо *ні*, і вона стане листком. Аналогічно, вершину 7 позначимо *так*, і вона також стане листком. Видучимо атрибут *температура* з множини *A*. Одержимо множину *A*={*вологість*, *вітер*}. Вершину 6 позначимо атрибутом *вологість*.

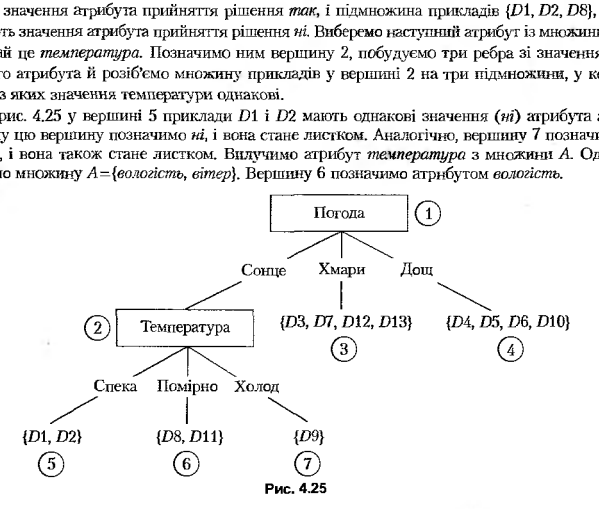


Рис. 4.25

На рис. 4.26 показано два листки, які відповідають значенням *так* і *ні* відповідно для прикладів *D*11 і *D*8. Видучимо атрибут *вологість* із множини *A* й отримаємо *A*={*вітер*}. Тепер розглянемо вершину 3 на рис. 4.24. Усі приклади, приписані цій вершині, мають

значення *так* атрибута *гра*, тому ця вершина являє собою листок; позначимо її *так*. У вершині 4 приклади мають різні значення атрибута *гра*, тому припишемо їй атрибут *вітер*, який має два значення: *слабкий* і *сильний*. Ці значення припишемо ребрам, інцидентним вершині 4. Множину прикладів розіб'ємо на дві підмножини: {*D*4, *D*5, *D*10} зі значенням *так* і {*D*6} зі значенням *ні*. Остаточно дерево рішень набуде вигляду, зображеного на рис. 4.27.

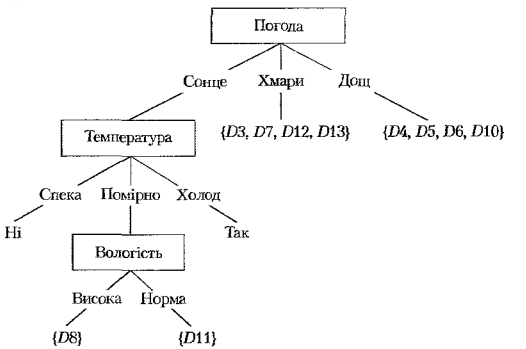


Рис. 4.26

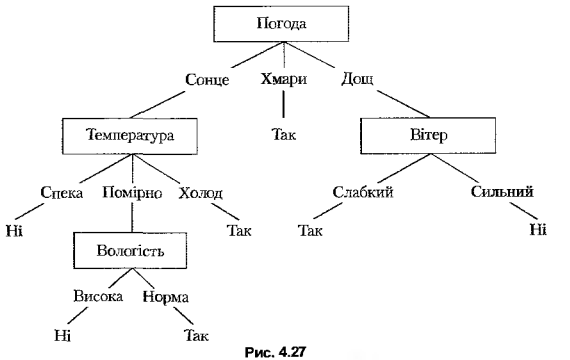


Рис. 4.27

Приклад 4.14. Вибираючи іншу послідовність атрибутів і обходу вершин, можна одержати інше дерево рішень. Дерево рішень, наведене на рис. 4.28, має менше вершин (немає атрибута *температура*). Для нього множина правил має такий вигляд:

```
If (погода=сонце) ∧ (вологість=висока) then (гра=ні);
If (погода=сонце) ∧ (вологість=норма) then (гра=так).
```

```
If (погода=хмари) then (гра=так);
If (погода=дощ) ∧ (вітер=слабкий) then (гра=так);
If (погода=дощ) ∧ (вітер=сильний) then (гра=ні).
```

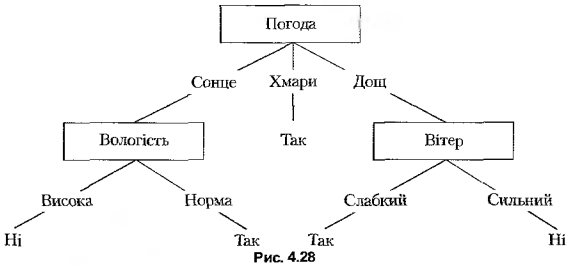


Рис. 4.28

Отже, за наявними прикладами будують дерево рішень. На його основі виписують множину правил, використовуваних для прийняття рішень у нових прикладах.

Порівнявши приклади 4.13 і 4.14, зазначимо, що на практиці реалізують версію алгоритму ID3, у якій аналізують важливість певних атрибутів для прийняття рішення. У такому разі під час вибору кореня поточного піддерева оцінюють частку інформації, додаваної кожним атрибутом. Потім вибирають той атрибут, який надає найбільше нової інформації.

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

4.5. Бектрекінг (пошук із поверненнями)

Опишемо загальний метод, який дає змогу значно зменшити обсяг обчислень в алгоритмах типу повного перебору всіх можливостей. Щоб застосувати цей метод, розв'язок задачі повинен мати вигляд скінченної послідовності $(x_1, ..., x_n)$. Головна ідея методу полягає в тому, що розв'язок будують поступово, починаючи з порожньої послідовності λ (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 одержання повного розв'язку. Якщо таке допустиме, але ще не використане значення x_{i+1} існує, то долучаємо цю нову компоненту до часткового розв'язку та 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, іще не використане значення x'_i. Тому процес називають *бектрекінгом* (англ. backtracking — пошук із поверненнями).

Роботу цього алгоритму можна інтерпретувати як процес обходу якогось дерева. Кожна його вершина відповідає якійсь послідовності $(x_1, ..., x_i)$, причому верши-

ни, які відповідають послідовностям вигляду $(x_1, ..., x_i, y)$, – сини цієї вершини. Корінь дерева відповідає порожній послідовності.

Виконується обхід цього дерева попуком углиб. Орім того, задають предикат P, означений на всіх вершинах дерева. Якщо $P(v)=$ F, то вершини піддерева з коренем у вершині v не розглядають, і обсяг перебору зменшується. Предикат $P(v)$ набуває значення F 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 бектрекінг на конкретних прикладах.

Приклад 4.15. Побудова гамільтонових циклів у графі [23]. Починаємо з довільної вершини. Будуємо шлях без повторення вершин, доки це можливо. Якщо вдалося пройти всі вершини, то перевіряємо, чи існує ребро, що з'єднує останню й початкову вершини цього шляху. Якщо описаний процес у певний момент неможливо продовжити, то повертаємося на одну вершину назад і намагаємося продовжити побудову шляху (без повторення вершин) іншим способом.

Пошук усіх гамільтонових циклів у графі з п'ятьма вершинами (рис. 4.29) можна проілюструвати за допомогою дерева, зображеного на рис. 4.30. Роботу алгоритму почато з одноелементної послідовності, бо в циклі вибір першої вершини неістотний. Розв'язки задачі обведено, і до них у прямокутних рамках приписано відповідні гамільтонові цикли.

Отже, замість побудови й аналізу $5!=120$ послідовностей довжиною 5 вершин графа, зображеного на рис. 4.28, ми розглянули лише 23 послідовності довжиною від 1 до 5.

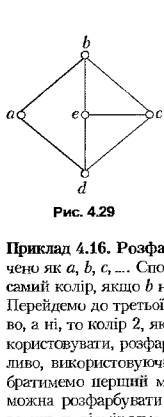


Рис. 4.29

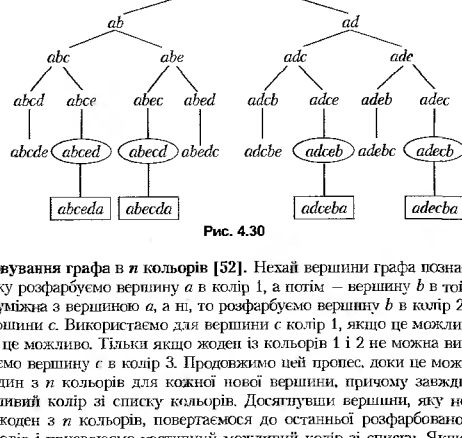


Рис. 4.30

Приклад 4.16. Розфарбовування графа в n кольорів [52]. Нехай вершини графа позначено як $a, b, c, ...$. Спочатку розфарбуємо вершину a в колір 1, а потім – вершину b в той самий колір, якщо b не суміжна з вершиною a, а ні, то розфарбуємо вершину b в колір 2. Перейдемо до третьої вершини c. Використаємо для вершини c колір 1, якщо це можливо, а ні, то колір 2, якщо це можливо. Тільки якщо жоден із кольорів 1 і 2 не можна використовувати, розфарбуємо вершину c в колір 3. Продовжимо цей процес, доки це можливо, використовуючи один з n кольорів для кожної нової вершини, причому завжди братимемо перший можливий колір зі списку кольорів. Досягнувши вершини, яку не можна розфарбувати в жоден з n кольорів, повертаємося до останньої розфарбованої вершини, відміняємо її колір і присвоюємо наступний можливий колір зі списку. Якщо

й це неможливо, то ще раз повертаємося до попередньої вершини, відміняємо її колір і намагаємося присвоїти новий колір, наступний можливий зі списку. Цей процес продовжуємо аналогічно. Якщо розфарбування в n кольорів існує, то така процедура дає змогу знайти його. На рис. 4.31 зображено граф і процес присвоювання трьох кольорів вершинам із використанням алгоритму бектрекінг.

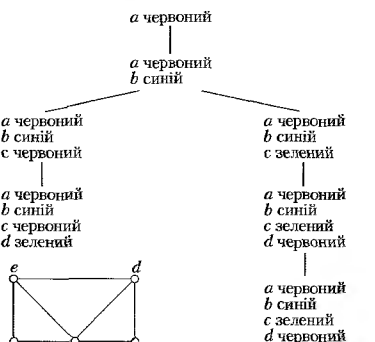


Рис. 4.31

Приклад 4.17. Задача про n ферзів [7, 52]. Як n ферзів можна розмістити на шахівниці $n \times n$ так, щоб жодні два 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 якогось 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 в перших k стовпцях уже 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 рядок є, а ні, то повертаємося до ферзя в $(k-1)$-му стовпці. Алгоритм бектрекінг для $n=4$ проілюстровано на рис. 4.32.

Кожній вершині дерева на рис. 4.32 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: λ, (2), (2, 4). (2. 4, 1), (2. 4. 1, 3).

Приклад 4.18. Суми елементів підмножин [52]. Задано множину натуральних чисел $\{x_1, x_2, ..., x_n\}$. Потрібно знайти її підмножину, сума елементів якої дорівнює заданому числу M.

Починаємо з порожньої множини. Нагромаджуємо суму, послідовно добираючи доданки. Число з послідовності $x_1, x_2, ..., x_n$ 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. 4.33 проілюстровано алгоритм бектрекінг для задачі відшукання підмножини множини {31, 27, 15, 11, 7, 5} із сумою 39.

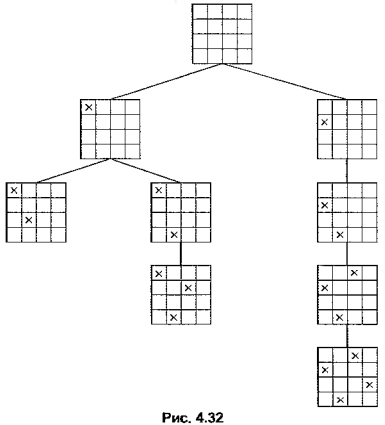


Рис. 4.32

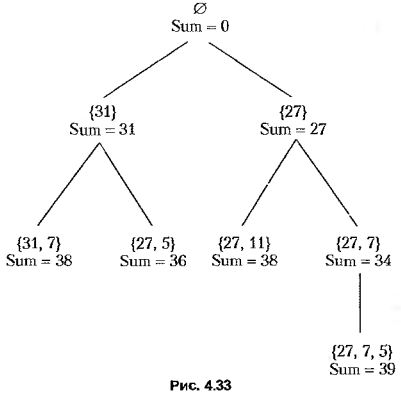


Рис. 4.33

Приклад 4.19. Побудова всіх максимальних незалежних множин вершин у простому графі $G=(V, \Gamma)$ [20, 35]. Тут граф подано як пару, утворену множиною вершин V та

відповідністю Г, котра показує, як зв'язані між собою вершини (див. підрозділ 3.3). Почнемо з порожньої множини й будемо додавати до неї вершини зі збереженням незалежності. Нехай S_k — уже отримана множина з k вершин, Q_k — множина вершин, котрі можна додати до S_k, тобто $S_k \cap \Gamma(Q_k) = \varnothing$. Серед вершин Q_k розрізняють ті, котрі вже було використано для розширення S_k (їх позначають Q_k^-), і ті, котрі ще не використано (їх позначають Q_k^+). Загальна схема алгоритму бектрекінг для задачі побудови максимальних незалежних множин вершин у простому графі має такий вигляд.

Прямий крок від k до $k+1$ полягає у виборі вершини $x \in Q_k^+$:

$$S_{k+1} := S_k \cup \{x\}; \qquad Q_{k+1}^- := Q_k^- \setminus \Gamma(x); \qquad Q_{k+1}^+ := Q_k^+ \setminus (\Gamma(x) \cup \{x\}).$$

Крок повернення від $k+1$ до k:

$$S_k := S_{k+1} \setminus \{x\}; \qquad Q_k^+ := Q_k^+ \setminus \{x\}; \qquad Q_k^- := Q_k^- \cup \{x\}.$$

Якщо множина вершин S_k максимальна, то $Q_k^+ = \varnothing$. Якщо $Q_k^- \neq \varnothing$, то множину S_k було розширено раніше, і вона не максимальна. Отже, перевірку максимальності задають такою умовою: $Q_k^+ = Q_k^- = \varnothing$.

Доцільно намагатися почати кроки повернення якомога раніше, бо це обмежить розміри „непотрібної" частини дерева пошуку. У зв'язку з цим зауважимо таке. Нехай $v \in Q_k^-$ і $\Gamma(v) \cap Q_k^+ = \varnothing$. Цю вершину неможливо вилучить з Q_k^-, оскільки можна вилучати лише вершини, суміжні з вершинами множини Q_k^+. Отже, існування такої вершини v, що $v \in Q_k^-$ і $\Gamma(v) \cap Q_k^+ = \varnothing$ — достатня умова для повернення. Окрім того, $k \leqslant n-1$.

Алгоритм побудови всіх максимальних незалежних множин вершин у простому графі $G = (V, \Gamma)$

Наведемо кроки алгоритму.

Крок 1. Ініціалізація. Виконати $S_0 := \varnothing$, $Q_0^- := \varnothing$, $Q_0^+ := V$, $k = 0$.

Крок 2. Прямий крок. Вибрати вершину $x \in Q_k^+$ і побудувати, як описано вище, множини S_{k+1}, Q_{k+1}^-, Q_{k+1}^+; при цьому залишити Q_k^- та Q_k^+ без змін. Виконати $k := k+1$.

Крок 3. Перевірка. Якщо виконано умову $\exists v \in Q_k^- : \Gamma(v) \cap Q_k^+ = \varnothing$, то перейти до кроку 5, інакше до кроку 4.

Крок 4. Якщо $Q_k^+ = Q_k^- = \varnothing$, то надрукувати максимальну незалежну множину S_k та перейти до кроку 5. Якщо $Q_k^+ = \varnothing$, а $Q_k^- \neq \varnothing$, то перейти до кроку 5. Інакше перейти до кроку 2.

Крок 5. Крок повернення. Виконати $k := k-1$. Вилучити вершину x із S_{k+1}, щоб одержати S_k. Виправити Q_k^- та Q_k^+, для чого вилучити вершину x із множини Q_k^+ і долучити її до Q_k^-. Якщо $k = 0$ та $Q_k^+ = \varnothing$, то зупинитися (на цей момент уже буде надруковано всі максимальні незалежні множини). Інакше перейти до кроку 3.

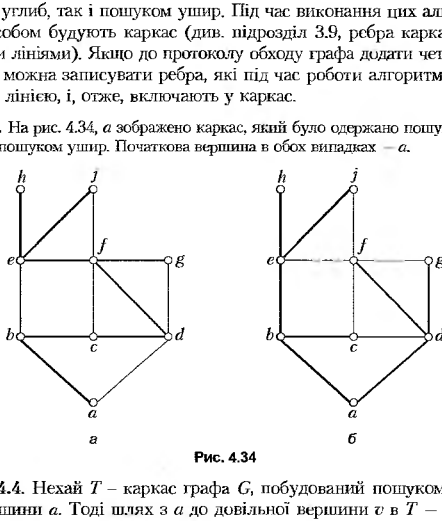
Унаслідок зв'язку між кліками й незалежними множинами (див. теорему 3.21) цей алгоритм можна використати також і для побудови максимальних клік.

4.6. Каркаси (з'єднувальні дерева)

Нехай G – простий зв'язний граф. *Каркасом*, або *з'єднувальним деревом*, графа G називають його підграф, який являє собою дерево та містить усі вершини графа G. Нехай граф G має n вершин і m ребер. Щоб отримати каркас, можна використати процедуру вилучення ребер, які належать простим циклам. Очевидно, що потрібно вилучити $\gamma(G)=m-(n-1)=m-n+1$ ребер. Число $\gamma(G)$ називають *цикломатичним числом* графа G. Із теореми 3.6 випливає, що $\gamma(G)\geqslant 0$. Цикломатичне число – це певною мірою числова характеристика зв'язності графа; цикломатичне число дерева дорівнює 0.

Побудова каркаса – поширена задача. Алгоритм, який полягає у вилученні ребер із простих циклів, неефективний для комп'ютерної реалізації. Для його виконання потрібно ідентифікувати прості цикли, а це складна задача. Ефективний із погляду комп'ютерної реалізації алгоритм побудови каркаса – це послідовний добір ребер у каркас. Це можна зробити за допомогою обходу графа G як пошуком углиб, так і пошуком ушир. Під час виконання цих алгоритмів природним способом будують каркас (див. підрозділ 3.9, ребра каркаса позначено потовщеними лініями). Якщо до протоколу обходу графа додати четвертий стовпчик, то туди можна записувати ребра, які під час роботи алгоритму позначають потовщеною лінією, і, отже, включають у каркас.

Приклад 4.20. На рис. 4.34, *а* зображено каркас, який було одержано пошуком углиб, а на рис. 4.34, *б* – пошуком ушир. Початкова вершина в обох випадках – a.



а *б*

Рис. 4.34

ТЕОРЕМА 4.4. Нехай T – каркас графа G, побудований пошуком ушир, починаючи з вершини a. Тоді шлях з a до довільної вершини v в T – найкоротший шлях від a до v в графі G.

Доведення ґрунтується на аналізі алгоритму пошуку вшир у графі G. Пропонуємо виконати його як вправу.

Зауваження. Використавши для побудови дерева найкоротших шляхів від вершини a алгоритм Дейкстри (уважаємо, що довжина кожного ребра дорівнює 1),

отримаємо те саме дерево, що й за алгоритмом пошуку вшир. Проте складність цього алгоритму становить $O(n^2)$ чи $O(m \log n)$ залежно від способу подання графа, а складність алгоритму пошуку вшир — $O(m+n)$, де n — кількість вершин, m — кількість ребер у графі G. Проте алгоритм Дейкстри більш універсальний, він придатний для будь-яких додатних довжин ребер (а не тільки 1).

Тепер розглянемо одну важливу задачу, пов'язану з ідеєю оптимізації. Нехай G — зв'язний зважений граф (див. підрозділ 3.8). Потрібно описати алгоритм побудови каркаса T, у якому сума ваг ребер $M(T)=\Sigma w(e)$ якнайменша (суму Σ беруть за всіма ребрами дерева T). Каркас T називають *мінімальним*. Розв'язати цю задачу дає змогу *алгоритм Краскала* (J. Kruskal).

Алгоритм Краскала

Нехай G — зв'язний зважений граф з n вершинами та m ребрами. Виконати такі дії.

1. Вибрати ребро e_1, яке має в графі G найменшу вагу.
2. Визначити послідовність ребер $e_2, e_3, ..., e_{n-1}$; на кожному кроці вибирати відмінне від попередніх ребро з найменшою вагою й таке, що не утворює простих циклів з уже вибраними ребрами. Одержане дерево T з множиною ребер $ET=\{e_1, e_2, e_3, ..., e_{n-1}\}$ — мінімальний каркас графа G.

Розглянемо одну з можливих реалізацій алгоритму Краскала.

Крок 1. Упорядкувати множину ребер за зростанням ваг: $e_1, e_2, e_3, ..., e_m$.

Крок 2. Утворити розбиття множини вершин на одноелементні підмножини: $\{\{v_1\}, \{v_2\}, ..., \{v_n\}\}$.

Крок 3. Вибирати таке чергове ребро з упорядкованої послідовності ребер, що його кінці містяться в різних множинах розбиття (це забезпечить відсутність простих циклів). Якщо вибрано ребро $e_i=\{v, w\}$, то множини розбиття, які містять вершини v та w, об'єднують в одну множину.

Крок 4. Якщо вже вибрано $(n-1)$ ребро (у такому разі всі підмножини розбиття об'єднаються в одну), то зупинитись, бо вибрані ребра утворюють мінімальний каркас. Інакше перейти до кроку 3.

Приклад 4.21. На рис. 4.35 зображено граф, ребра якого пронумеровано за зростанням ваг. Мінімальний каркас виділено потовщеними лініями. Процес відбору ребер наведено в табл. 4.4. Мінімальний каркас утворюють ребра $e_1, e_2, e_3, e_5, e_8, e_{11}$.

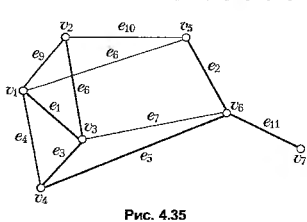


Рис. 4.35

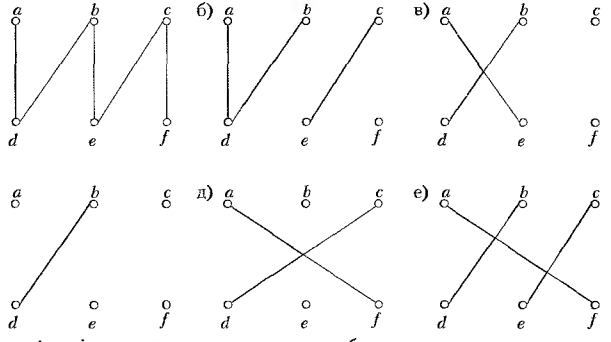
Таблиця 4.4

Ребро	Розбиття множини вершин	Відбір ребра у мінімальний каркас
$e_1=\{v_1,v_3\}$	$\{\{v_1\}, \{v_2\}, \{v_3\}, \{v_4\}, \{v_5\}, \{v_6\}, \{v_7\}\}$	e_1
$e_2=\{v_5,v_6\}$	$\{\{v_1,v_3\}, \{v_2\}, \{v_4\}, \{v_5\}, \{v_6\}, \{v_7\}\}$	e_2
$e_3=\{v_3,v_4\}$	$\{\{v_1,v_3\}, \{v_2\}, \{v_4\}, \{v_5,v_6\}, \{v_7\}\}$	e_3
$e_4=\{v_1,v_4\}$	$\{\{v_1,v_3,v_4\}, \{v_2\}, \{v_5,v_6\}, \{v_7\}\}$	–
$e_5=\{v_4,v_6\}$	$\{\{v_1,v_3,v_4\}, \{v_2\}, \{v_5,v_6\}, \{v_7\}\}$	e_5
$e_6=\{v_1,v_5\}$	$\{\{v_1,v_3,v_4,v_5,v_6\}, \{v_2\}, \{v_7\}\}$	–
$e_7=\{v_3,v_6\}$	$\{\{v_1,v_3,v_4,v_5,v_6\}, \{v_2\}, \{v_7\}\}$	–
$e_8=\{v_2,v_3\}$	$\{\{v_1,v_3,v_4,v_5,v_6\}, \{v_2\}, \{v_7\}\}$	e_8
$e_9=\{v_1,v_2\}$	$\{\{v_1,v_2,v_3,v_4,v_5,v_6\}, \{v_7\}\}$	–
$e_{10}=\{v_2,v_5\}$	$\{\{v_1,v_2,v_3,v_4,v_5,v_6\}, \{v_7\}\}$	–
$e_{11}=\{v_6,v_7\}$	$\{\{v_1,v_2,v_3,v_4,v_5,v_6\}, \{v_7\}\}$	e_{11}
	$\{\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}\}$	

Алгоритм Краскала належить до *жадібних* [23]. Так називають алгоритми оптимізації, які на кожному кроці вибирають найкращий із можливих варіантів.

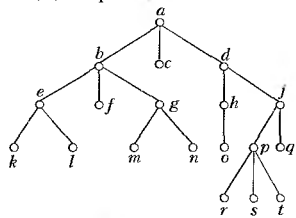
Контрольні запитання та завдання

1. Які з наведених нижче графів являють собою дерева?



2. Дати відповідь на запитання щодо дерева, зображеного на рисунку.
   а) Яка вершина являє собою корінь?
   б) Які вершини внутрішні?
   в) Які вершини являють собою листки?
   г) Які вершини – сини вершини j?

д) Яка вершина — батько вершини n?
е) Які вершини — предки вершини n?
ж) Які вершини — нащадки вершини b?

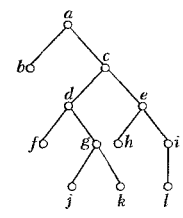


3. Визначити рівень кожної вершини та висоту дерева, зображеного на рисунку.
4. Для яких значень $m > 0$ й $n > 0$ повний дводольний граф $K_{m,n}$ являє собою дерево?
5. Скільки ребер має дерево з 1000 вершинами?

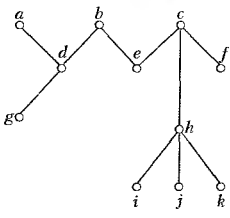
У задачах 6–10 використати теорему 4.2.

6. Скільки вершин має повне 5-арне дерево зі 100 внутрішніми вершинами?
7. Скільки ребер має повне бінарне дерево з 1000 внутрішніх вершин?
8. Скільки листків має повне 3-арне дерево зі 100 вершинами?
9. У шаховому турнірі беруть участь 1000 гравців. Скільки ігор потрібно зіграти для визначення переможця, якщо турнір проводять за олімпійською системою (той, хто програв, вибуває)?
10. Чи існує повне m-арне дерево T, яке має 84 листки та висоту 3?
11. Побудувати завершене бінарне дерево висотою 4 та завершене 3-арне дерево висотою 3.
12. Довести, що завершене m-арне дерево висотою h має $(m^{h+1} - 1)/(m - 1)$ вершин і m^h листків.
13. *Ексцентриситет* вершини в некореневому дереві — це довжина найдовшого простого шляху, який починається в цій вершині. Вершину називають *центром*, якщо вона має найменший ексцентриситет. Знайти центри дерев:

а)

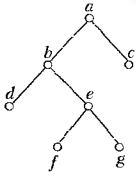


б)

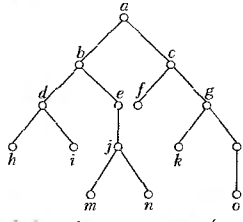


14. Довести, що центр у некореневому дереві можна знайти як корінь кореневого дерева з мінімальною висотою.
15. Довести, що дерево має один або два центри, суміжні між собою. Навести приклад дерева, яке має два центри.
16. Записати послідовності вершин упорядкованих кореневих дерев, наведених нижче, в разі їх обходу в прямому, зворотному та внутрішньому порядку.

а)



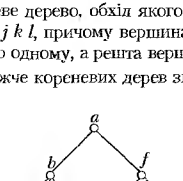
б)



17. Побудувати кореневі дерева, які відповідають виразам $(x + x * y) + (x/y)$ та $x + ((x * y + x)/y)$. Записати ці вирази в інфіксній, префіксній та постфіксній формах.
18. Побудувати кореневі дерева, які відповідають виразам $(\neg(p \wedge q)) \sim ((\neg p) \vee (\neg q))$ та $((\neg p) \wedge (q \sim (\neg p))) \vee (\neg q)$. Записати ці вирази в інфіксній, префіксній та постфіксній формах.
19. Зобразити впорядковане кореневе дерево, яке відповідає кожному з виразів, записаних у префіксній формі. Записати їх у постфіксній формі:
а) ↑ + 2 3 – 5 1; б) + * + – 5 3 2 1 4; в) * / 9 3 + * 2 4 – 7 6.
20. Обчислити значення виразів, записаних у префіксній формі:
а) – * 2 / 8 4 3; б) ↑ – * 3 3 * 4 2 5;
в) + – ↑ 3 2 ↑ 2 3 / 6 –4 2; г) * + 3 + 3 ↑ 3 + 3 3 3.
21. Обчислити значення виразів, записаних у постфіксній формі:
а) 9 3 / 5 + 7 2 – *; б) 5 2 1 – – 3 1 4 + + *; в) 3 2 * 2 ↑ 5 3 – 8 4 / * –.
22. Побудувати впорядковане кореневе дерево, обхід якого в прямому порядку дає послідовність *a b f c g h i d e j k l*, причому вершина *a* має чотирьох синів, *c* – трьох, *j* – двох, *b* й *e* – по одному, а решта вершин – листки.
23. Показати, що обхід наведених нижче кореневих дерев зверху вниз дає однакові списки вершин.

а)

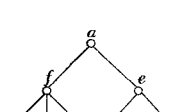
б)



24. Показати, що обхід наведених нижче кореневих дерев знизу вверх дає однакові списки вершин.

а)

б)



У задачах 25, 26 доведення виконати методом математичної індукції.

25. Довести, що впорядковане кореневе дерево можна однозначно задати списком вершин, одержаним обходом у прямому порядку, якщо для кожної вершини зазначити кількість її синів.

26. Довести, що впорядковане кореневе дерево можна однозначно задати списком вершин, отриманим обходом у зворотному порядку, якщо для кожної вершини зазначити кількість її синів.

27. Побудувати бінарне дерево пошуку для слів *banana, peach, pear, apple, coconut, mango, papaya.*

28. Скільки потрібно порівнянь, щоб знайти чи додати кожне зі слів до дерева пошуку із задачі 27:

а) *pear*; б) *banana*;

в) *plum*; г) *orange*.

29. Використати бектрекінг для відшукання гамільтонових циклів у графах із задачі 50 розділу 3.

30. Використовуючи бектрекінг, розфарбувати в три кольори графи із задач 73 і 70 розділу 3.

31. Використовуючи бектрекінг, розв'язати задачу про n ферзів для значень n, що дорівнюють 3, 5 і 6.

32. Використати бектрекінг для відшукання всіх максимальних незалежних множин вершин графів із задачі 74 розділу 3.

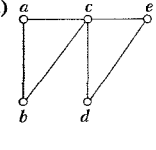
33. Звести задачу про n ферзів до задачі про найбільшу незалежну множину вершин у графі. Проілюструвати для значень n, що дорівнюють 3 та 4.

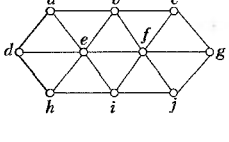
34. Використовуючи бектрекінг, знайти підмножину (якщо вона існує) множини {27, 24, 19, 14, 11, 8} із зазначеною сумою:

а) 20; б) 41; в) 60.

35. Зв'язний граф G має n вершин і m ребер. Скільки ребер потрібно вилучити з графа G, щоб одержати каркас?

36. Знайти каркас для кожного з наведених нижче графів способом вилучення ребер із простих циклів.

а)


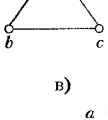
б)


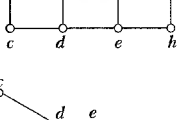
в)

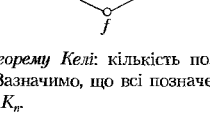

37. Побудувати каркаси для кожного з наведених нижче графів. Знайти цикломатичне число кожного графа:

а) K_5; б) $K_{1,6}$; в) C_5; г) $K_{4,4}$; д) Q_3; е) W_5.

38. Для простих графів побудувати всі можливі каркаси.

а)


б)


в)


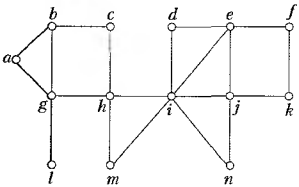
39. Довести *теорему Келі*: кількість позначених дерев з n вершинами дорівнює n^{n-2}. Зазначимо, що всі позначені дерева з n вершинами — це всі каркаси графа K_n.

40. Зобразити всі позначені дерева з n вершинами для значень n, що дорівнюють 2, 3 та 4.

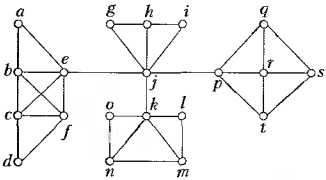
41. Скільки існує неізоморфних дерев з n вершинами для значень n, що дорівнюють 3, 4 та 5? Зобразити всі ці дерева.

42. Використовуючи обхід графа пошуком углиб, побудувати каркаси для графів, наведених нижче. Як початкову вибрати вершину *a*.

а)



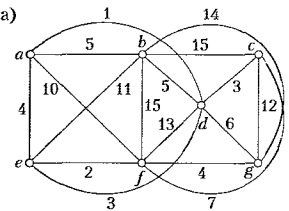
б)



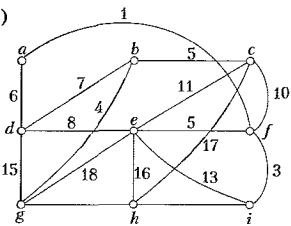
43. Розв'язати задачу 42, використовуючи пошук ушир.

44. За алгоритмом Краскала для наведених нижче графів побудувати мінімальний каркас.

а)



б)



Комп'ютерні проекти

Скласти програми із зазначеними вхідними даними та результатами.

1. Задано матрицю суміжності простого графа. Визначити, чи є цей граф деревом.
2. Простий граф задано списками суміжності. Визначити, чи є цей граф деревом.
3. Задано матрицю суміжності кореневого дерева та якусь його вершину. Знайти батька, синів, предків, нащадків і рівень цієї вершини.

4. Задано список ребер кореневого дерева та якусь його вершину. Знайти батька, синів, предків, нащадків і рівень цієї вершини.
5. Задано список об'єктів. Побудувати бінарне дерево пошуку, що містить ці об'єкти.
6. Задано бінарне дерево пошуку й об'єкт. Локалізувати цей об'єкт у дереві пошуку чи додати його до дерева.
7. Задано список ребер некореневого дерева. Знайти центри цього дерева (див. задачу 13).
8. Задано впорядкований список ребер упорядкованого кореневого дерева. Записати його вершини в разі обходу в прямому, зворотному та внутрішньому порядку.
9. Задано арифметичний вираз у польському записі. Обчислити його значення.
10. Задано арифметичний вираз у зворотному польському записі. Обчислити його значення.
11. Задано матрицю суміжності простого зв'язного графа. Побудувати каркас цього графа з використанням пошуку вглиб.
12. Простий зв'язний граф задано списками суміжності. Побудувати каркас цього графа з використанням пошуку вглиб.
13. Задано матрицю суміжності простого зв'язного графа. Побудувати каркас цього графа з використанням пошуку вшир.
14. Простий зв'язний граф задано списками суміжності. Побудувати каркас цього графа з використанням пошуку вшир.
15. Задано скінченну множину натуральних чисел і натуральне число n. Використовуючи бектрекінг, знайти таку підмножину цих чисел, щоб їх сума дорівнювала n.
16. Задано матрицю суміжності простого зв'язного графа. Побудувати гамільтонів цикл або виявити, що даний граф не має такого циклу. Використати бектрекінг.
17. Простий зв'язний граф задано списками суміжності. Побудувати гамільтонів цикл або виявити, що даний граф не має такого циклу. Використати бектрекінг.
18. Задано матрицю суміжності простого зв'язного графа. Побудувати гамільтонів шлях або виявити, що даний граф не має його. Використати бектрекінг.
19. Простий зв'язний граф задано списками суміжності. Побудувати гамільтонів шлях або виявити, що даний граф не має його. Використати бектрекінг.
20. Задано матрицю суміжності простого зв'язного графа та натуральне число n. Використовуючи бектрекінг, розфарбувати цей граф у n кольорів або переконатись, що це неможливо.
21. Задано списки суміжності простого зв'язного графа та натуральне число n. Використовуючи бектрекінг, розфарбувати цей граф у n кольорів або переконатись, що це неможливо.

22. Задано натуральне число n. Використовуючи бектрекінг, розв'язати задачу про n ферзів.
23. Задано шахівницю $n \times n$. Кінь, який ходить за шаховими правилами, перебуває в клітинці з початковими координатами (i_0, j_0), де i_0 — номер рядка, j_0 — номер стовпця. Визначити, чи можна покрити всю шахівницю за n^2-1 ходів коня так, щоб кожну клітинку було відвідано точно один раз. Використати бектрекінг.
24. Задано матрицю суміжності простого зв'язного графа. Знайти всі максимальні незалежні множини вершин цього графа. Використати бектрекінг.
25. Простий зв'язний граф задано списками суміжності. Знайти всі максимальні незалежні множини вершин цього графа. Використати бектрекінг.
26. Задано матрицю суміжності простого зв'язного графа. Знайти всі максимальні кліки. Використати бектрекінг.
27. Простий зв'язний граф задано списками суміжності. Знайти всі максимальні кліки. Використати бектрекінг.
28. Зважений простий зв'язний граф задано матрицею ваг. Використовуючи алгоритм Краскала, побудувати мінімальний каркас цього графа.
29. Задано список ребер та їх ваг для зваженого простого зв'язного графа. За допомогою алгоритму Краскала побудувати мінімальний каркас цього графа.