Contents

1 Polynomial Time **2**

2 Polynomial Time Verification **2**

3 Class NP **2**

4 Reducibility **3**

5 NP-Complete Problems **3**

6 How to Prove a Problem is NP-Complete **3**

7 Common NP-Complete Problems **4**

8 Summary Table **4**

9 Key Exam Points **5**

10 Two Circle Intersection **5**
- 10.1 Cases . . . 6
- 10.2 Python Implementation . . . 7
- 10.3 Illustrations . . . 8

11 Check if a point is inside or outside of a polygon **9**

12 Quardic hashing proof **9**

13 28 Mid **11**

14 27 Mid **13**

15 Final 27 **15**
- 15.1 qs 2 . . . 15

16 Qs 3 **15**
- 16.1 Qs 4 . . . 16

1 Polynomial Time

Definition: An algorithm is said to run in **polynomial time** if its running time is

$$O(n^k)$$

for some constant k, where n is the size of the input.

Examples:

- Sorting: $O(n \log n)$
- Matrix multiplication: $O(n^3)$
- Graph BFS/DFS: $O(V + E)$

Class P: The class **P** consists of all decision problems that can be solved in polynomial time by a deterministic Turing machine.

$$\mathbf{P} = \{L \mid L \text{ is solvable in polynomial time}\}$$

—

2 Polynomial Time Verification

Definition: A problem has **polynomial-time verification** if, given a proposed solution (certificate), we can verify its correctness in polynomial time.

Example: Hamiltonian Cycle

- Certificate: A sequence of vertices
- Verification: Check if every vertex appears once and edges exist
- Time: Polynomial

Key Idea:

"Checking a solution is easier than finding it."

—

3 Class NP

Definition: The class **NP** consists of all decision problems for which a given solution can be verified in polynomial time.

$$\mathbf{NP} = \{L \mid L \text{ has polynomial-time verification}\}$$

Equivalent Definition:

- Solvable by a nondeterministic Turing machine in polynomial time

Relationship:

$$\mathbf{P} \subseteq \mathbf{NP}$$

Whether $\mathbf{P} = \mathbf{NP}$ is an **open problem**.

—

4 Reducibility

Polynomial-Time Reduction (Many-One Reduction):

A problem A is polynomial-time reducible to problem B, written

$$A \leq_p B$$

if any instance of A can be transformed into an instance of B in polynomial time such that:

$$A \text{ is YES} \iff B \text{ is YES}$$

Purpose of Reduction:

- Compare difficulty of problems
- Prove NP-completeness

Important Property:

If $A \leq_p B$ and $B \in \mathbf{P}$, then $A \in \mathbf{P}$.

—

5 NP-Complete Problems

Definition: A problem L is **NP-complete** if:

(i) $L \in \mathbf{NP}$

(ii) Every problem in **NP** is reducible to L in polynomial time

$$\mathbf{NPC} = \{L \mid L \in \mathbf{NP} \text{ and NP-hard}\}$$

Meaning:

- Hardest problems in NP
- If any NP-complete problem is in P, then $P = NP$

—

6 How to Prove a Problem is NP-Complete

1. Show the problem is in NP
2. Choose a known NP-complete problem
3. Reduce the known problem to the given problem in polynomial time

$$\text{Known NPC} \ \leq_p \ \text{New Problem}$$

—

7 Common NP-Complete Problems

- SAT (Boolean Satisfiability Problem)
- 3-SAT
- Clique
- Vertex Cover
- Hamiltonian Cycle
- Traveling Salesman Problem (Decision version)
- Subset Sum
- Knapsack (Decision version)

Cook–Levin Theorem: SAT was the first problem proven to be NP-complete.
—

8 Summary Table

Class	**Meaning**	**Example**
P	Problems that can be solved in polynomial time	Sorting, Shortest path (BFS)
NP	Problems whose solutions can be verified in polynomial time	Hamiltonian Cycle (verification)
NP-Hard	Problems at least as hard as NP problems, not necessarily in NP	Optimization version of TSP
NP-Complete	Problems that are in NP and NP-Hard	SAT, 3-SAT, Vertex Cover

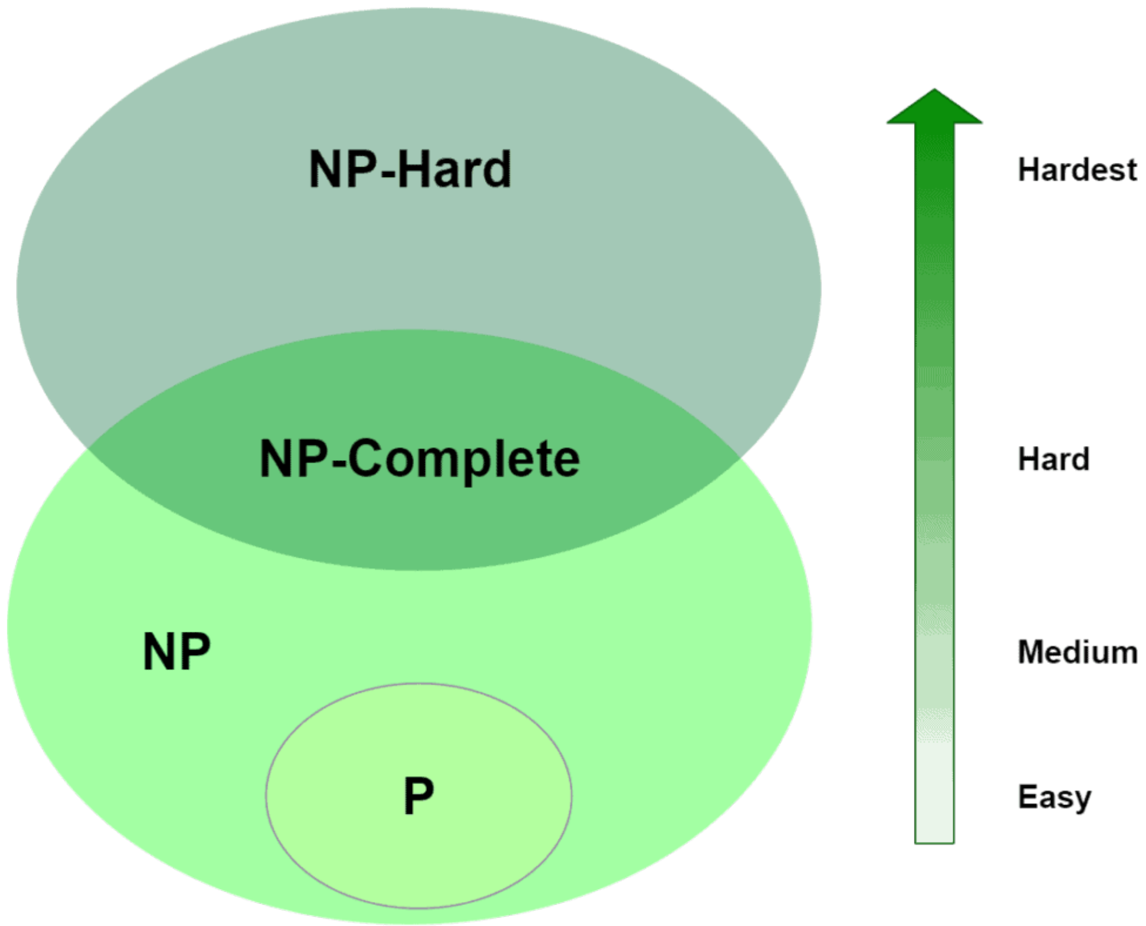


Figure 1: Caption

9 Key Exam Points

- P vs NP is unknown
- NP-complete problems are decision problems
- Reduction direction is very important
- Verification time defines NP, not solving time

10 Two Circle Intersection

The distance between the centers $C_1(x_1, y_1)$ and $C_2(x_2, y_2)$ is:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

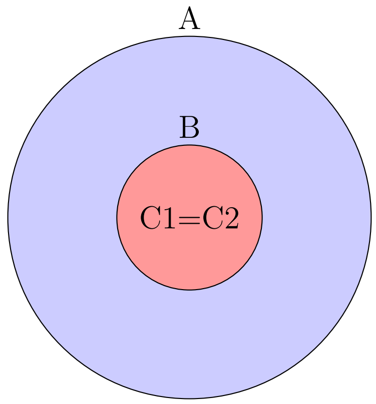
10.1 Cases

$$\begin{aligned}
&\text{If } d \leq R_1 - R_2 && \Rightarrow \text{ Circle B is inside A.}\\
&\text{If } d \leq R_2 - R_1 && \Rightarrow \text{ Circle A is inside B.}\\
&\text{If } d < R_1 + R_2 && \Rightarrow \text{ Circles intersect.}\\
&\text{If } d = R_1 + R_2 && \Rightarrow \text{ Circles touch externally.}\\
&\text{If } d > R_1 + R_2 && \Rightarrow \text{ Circles do not overlap.}
\end{aligned}$$

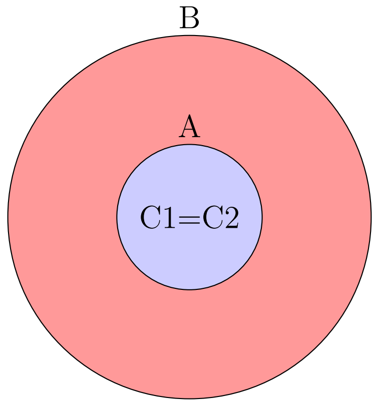
10.2 Python Implementation

```python
import math

# Function to check the relation between two circles
def circle_relation(x1, y1, r1, x2, y2, r2):
    """
    Determines the spatial relationship between two circles.
    """
    d = math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

    if d <= abs(r1 - r2):
        if r1 > r2:
            print("Circle B is inside Circle A")
        elif r2 > r1:
            print("Circle A is inside Circle B")
        else:
            print("Circles are coincident (identical)")
    elif d < r1 + r2:
        print("Circles intersect each other")
    elif d == r1 + r2:
        print("Circles touch each other externally")
    else:
        print("Circles do not overlap")

# Example usage
if __name__ == "__main__":
    circle_relation(-10, 8, 30, 14, -24, 10)
```

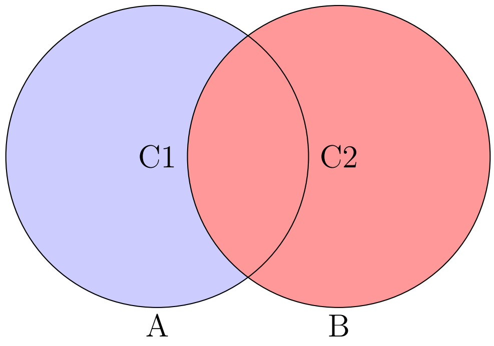
10.3 Illustrations



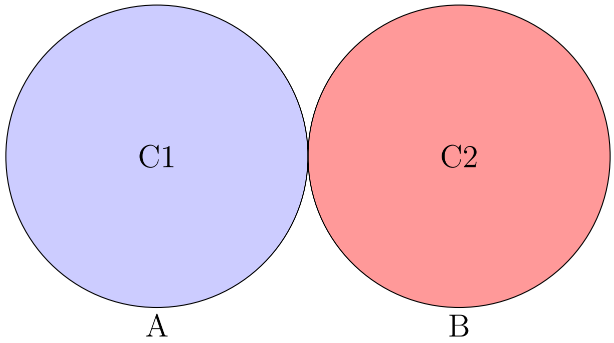
Case 1: Circle B inside A ($d \leq R_1 - R_2$).



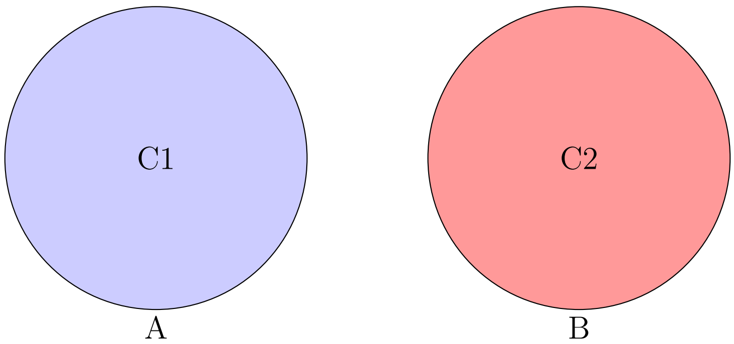
Case 2: Circle A inside B ($d \leq R_2 - R_1$).



Case 3: Circles intersect ($d < R_1 + R_2$).



Case 4: Circles touch externally ($d = R_1 + R_2$).



Case 5: Circles do not overlap ($d > R_1 + R_2$).

11 Check if a point is inside or outside of a polygon

Click here to visit Example click

12 Quardic hashing proof

Proof. Let m be the size of the hash table, and assume m is a prime number. The quadratic probing sequence for a key with a hash value $h'(k)$ is given by:

$$h(k, i) = (h'(k) + i^2) \pmod m \quad \text{for } i = 0, 1, 2, \dots$$

We need to prove that if the load factor $\alpha \le 0.5$, then an empty slot will always be found. This is guaranteed if the first $\lfloor m/2 \rfloor$ probes visit unique positions. The load factor $\alpha = \frac{N}{m}$, where N is the number of occupied slots. If $\alpha \le 0.5$, then $\frac{N}{m} \le 0.5$, which implies $N \le \lfloor m/2 \rfloor$. The number of empty slots is $m - N \ge m - \lfloor m/2 \rfloor = \lceil m/2 \rceil$.

We will now prove by contradiction that the first $\lfloor m/2 \rfloor$ probes are unique. Assume, for contradiction, that two different probe numbers, i and j, where $0 \le i < j \le \lfloor m/2 \rfloor$, map to the same slot.

$$(h'(k) + i^2) \pmod m = (h'(k) + j^2) \pmod m$$

Subtracting $h'(k)$ from both sides, we get:

$$i^2 \pmod m = j^2 \pmod m$$

This can be rewritten as:

$$(j^2 - i^2) \pmod m = 0$$

Factoring the difference of squares gives:

$$(j - i)(j + i) \pmod m = 0$$

Since m is a prime number, by Euclid's lemma, if m divides a product of two numbers, it must divide at least one of them. Thus, either $(j-i) \pmod m = 0$ or $(j+i) \pmod m = 0$.

Case 1: $(j - i) \pmod m = 0$ This means $j - i$ is a multiple of m. However, we know that $0 \le i < j \le \lfloor m/2 \rfloor$, so:

$$0 < j - i \le \lfloor m/2 \rfloor - 0 = \lfloor m/2 \rfloor$$

Since m is prime and $m \ge 2$, it follows that $\lfloor m/2 \rfloor < m$. Therefore, $0 < j - i < m$. A number in this range cannot be a multiple of m. This is a contradiction.

Case 2: $(j + i) \pmod m = 0$ This means $j + i$ is a multiple of m. We know that $0 \le i < j \le \lfloor m/2 \rfloor$, so:

$$0 < j + i \le \lfloor m/2 \rfloor + \lfloor m/2 \rfloor = 2\lfloor m/2 \rfloor$$

For any prime $m > 2$, m is odd, so $m = 2k + 1$ for some integer k. Then $\lfloor m/2 \rfloor = k$.

$$j + i \le 2\lfloor m/2 \rfloor = 2k < 2k + 1 = m$$

Therefore, $0 < j + i < m$. A number in this range cannot be a multiple of m. This is also a contradiction.

Since both cases lead to a contradiction, our initial assumption must be false. The first $\lfloor m/2 \rfloor$ probes must all land in unique slots. Because the load factor $\alpha \le 0.5$, the number of occupied slots $N \le \lfloor m/2 \rfloor$. The first $\lfloor m/2 \rfloor$ unique probes are guaranteed to check at least one empty slot. Therefore, an empty slot will always be found. $\square$

Uniqueness Part of the Chinese Remainder Theorem

Theorem (Chinese Remainder Theorem – Uniqueness). Let $m_1, m_2, \ldots, m_k$ be positive integers such that

$$\gcd(m_1, m_2, \ldots, m_k) = 1.$$

Then the system of congruences

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

has *at most one* solution modulo $m_1 m_2 \cdots m_k$.

Proof

Let X_1 and X_2 be any two solutions of the above system of congruences. Then for each i with $1 \le i \le k$, we have

$$X_1 \equiv a_i \pmod{m_i} \quad \text{and} \quad X_2 \equiv a_i \pmod{m_i}.$$

Subtracting the two congruences gives

$$X_1 \equiv X_2 \pmod{m_i},$$

which implies

$$m_i \mid (X_1 - X_2).$$

Since this holds for every $i = 1, 2, \ldots, k$, it follows that

$$\operatorname{lcm}(m_1, m_2, \ldots, m_k) \mid (X_1 - X_2).$$

Because $m_1, m_2, \ldots, m_k$ are pairwise relatively prime, their least common multiple is

$$\operatorname{lcm}(m_1, m_2, \ldots, m_k) = m_1 m_2 \cdots m_k.$$

Therefore,

$$m_1 m_2 \cdots m_k \mid (X_1 - X_2),$$

which implies

$$X_1 \equiv X_2 \pmod{m_1 m_2 \cdots m_k}.$$

Hence, any two solutions of the system are congruent modulo $m_1 m_2 \cdots m_k$. This proves that the system of congruences has *at most one* solution modulo $m_1 m_2 \cdots m_k$.

□

13 28 Mid

Q2. Hashing with Quadratic Probing

Given: Table size: $m = 4013$ (prime)

Hash function:

$$h(k) = (3k + 5) \bmod 4013$$

Quadratic probing:

$$h(k, i) = (h(k) + i^2) \bmod 4013$$

We need two consecutive probe indices i and $j = i + 1$, $i \neq j$, such that they produce the same table index.

Step 1: Condition for collision

$$h(k) + i^2 \equiv h(k) + (i+1)^2 \pmod{4013}$$

Cancel $h(k)$:

$$i^2 \equiv (i+1)^2 \pmod{4013}$$

Step 2: Simplify

$$(i+1)^2 - i^2 = 2i + 1$$

So we need:

$$2i + 1 \equiv 0 \pmod{4013}$$

Step 3: Solve

$$2i \equiv -1 \equiv 4012 \pmod{4013}$$

Since 4013 is prime, we can divide by 2:

$$i \equiv \frac{4012}{2} = 2006$$

$$j = i + 1 = 2007$$

$$\boxed{i = 2006,\ j = 2007}$$

These consecutive probe indices produce the same table index.

—

Q3. Convex Hull Area After Adding an Interior Point

Given: Finite set of points P, area of convex hull A.

Pick any two points $p_i, p_j \in P$, and define:

$$q = \lambda p_i + (1-\lambda)p_j, \quad 0 < \lambda < 1$$

Let:

$$P' = P \cup \{q\}, \quad B = \text{area of convex hull of } P'$$

Key Insight: q lies on the line segment between p_i and p_j, so q is inside or on the boundary of the convex hull of P.

Adding an interior/boundary point does not expand the convex hull.

$$\boxed{A = B \quad (\text{or equivalently, } A \leq B)}$$

—

Q4. Rabin–Karp and Maximum Alphabet Size

Given: Integer size = 32 bits, maximum pattern length $m = 5$, want linear-time Rabin–Karp.

Step 1: Rabin–Karp constraint

$$|\Sigma|^m \leq 2^{32}, \quad m = 5$$

Step 2: Solve inequality

$$|\Sigma|^5 \leq 2^{32} \implies |\Sigma| \leq \sqrt[5]{2^{32}} = 2^{32/5} \approx 84$$

$$\boxed{|\Sigma|_{\max} = 84}$$

14 27 Mid

Hash Table Insertion Problem

Given:

- Hash table length: $m = 11$
- Keys to insert: 10, 22, 31, 4, 15, 28, 17, 88, 59
- Auxiliary hash function: $h'(k) = k$

Method 1: Quadratic Probing with $C_1 = 1$ and $C_2 = 3$

Hash function:

$$h(k, i) = (h'(k) + C_1 \cdot i + C_2 \cdot i^2) \bmod m = (k + i + 3i^2) \bmod 11$$

Insertion Process:

1. **Key 10:** $h(10, 0) = 10 \bmod 11 = 10 \quad \checkmark$
2. **Key 22:** $h(22, 0) = 22 \bmod 11 = 0 \quad \checkmark$
3. **Key 31:** $h(31, 0) = 31 \bmod 11 = 9 \quad \checkmark$
4. **Key 4:** $h(4, 0) = 4 \bmod 11 = 4 \quad \checkmark$
5. **Key 15:**
$$h(15, 0) = 15 \bmod 11 = 4 \quad \text{(occupied)}$$
$$h(15, 1) = (15 + 1 + 3) \bmod 11 = 19 \bmod 11 = 8 \quad \checkmark$$
6. **Key 28:** $h(28, 0) = 28 \bmod 11 = 6 \quad \checkmark$
7. **Key 17:**
$$h(17, 0) = 17 \bmod 11 = 6 \quad \text{(occupied)}$$
$$h(17, 1) = (17 + 1 + 3) \bmod 11 = 21 \bmod 11 = 10 \quad \text{(occupied)}$$
$$h(17, 2) = (17 + 2 + 12) \bmod 11 = 31 \bmod 11 = 9 \quad \text{(occupied)}$$
$$h(17, 3) = (17 + 3 + 27) \bmod 11 = 47 \bmod 11 = 3 \quad \checkmark$$
8. **Key 88:**
$$h(88, 0) = 88 \bmod 11 = 0 \quad \text{(occupied)}$$
$$h(88, 1) = (88 + 1 + 3) \bmod 11 = 92 \bmod 11 = 4 \quad \text{(occupied)}$$
$$h(88, 2) = (88 + 2 + 12) \bmod 11 = 102 \bmod 11 = 3 \quad \text{(occupied)}$$
$$h(88, 3) = (88 + 3 + 27) \bmod 11 = 118 \bmod 11 = 8 \quad \text{(occupied)}$$
$$h(88, 4) = (88 + 4 + 48) \bmod 11 = 140 \bmod 11 = 8 \quad \text{(cycle)}$$

Key 88 cannot be inserted due to cycling.

Final Hash Table (Quadratic Probing):

Index	0	1	2	3	4	5	6	7	8	9	10
Key	22	-	-	17	4	-	28	-	15	31	10

Method 2: Double Hashing

Hash functions:

$$\begin{aligned} h_1(k) &= k \bmod 11 \\ h_2(k) &= 1 + (k \bmod (m-1)) = 1 + (k \bmod 10) \\ h(k,i) &= (h_1(k) + i \cdot h_2(k)) \bmod 11 \end{aligned}$$

Insertion Process:

1. **Key 10:** $h_1(10) = 10$, position $= 10$ ✓
2. **Key 22:** $h_1(22) = 0$, position $= 0$ ✓
3. **Key 31:** $h_1(31) = 9$, position $= 9$ ✓
4. **Key 4:** $h_1(4) = 4$, position $= 4$ ✓
5. **Key 15:** $h_1(15) = 4$ (occupied), $h_2(15) = 1 + 5 = 6$

$$\begin{aligned} h(15,1) &= (4+6) \bmod 11 = 10 \quad \text{(occupied)} \\ h(15,2) &= (4+12) \bmod 11 = 5 \quad \checkmark \end{aligned}$$

6. **Key 28:** $h_1(28) = 6$, position $= 6$ ✓
7. **Key 17:** $h_1(17) = 6$ (occupied), $h_2(17) = 1 + 7 = 8$

$$h(17,1) = (6+8) \bmod 11 = 3 \quad \checkmark$$

8. **Key 88:** $h_1(88) = 0$ (occupied), $h_2(88) = 1 + 8 = 9$

$$\begin{aligned} h(88,1) &= (0+9) \bmod 11 = 9 \quad \text{(occupied)} \\ h(88,2) &= (0+18) \bmod 11 = 7 \quad \checkmark \end{aligned}$$

9. **Key 59:** $h_1(59) = 4$ (occupied), $h_2(59) = 1 + 9 = 10$

$$\begin{aligned} h(59,1) &= (4+10) \bmod 11 = 3 \quad \text{(occupied)} \\ h(59,2) &= (4+20) \bmod 11 = 2 \quad \checkmark \end{aligned}$$

Final Hash Table (Double Hashing):

Index	0	1	2	3	4	5	6	7	8	9	10
Key	22	-	59	17	4	15	28	88	-	31	10

15 Final 27

15.1 qs 2

Linear Time Algorithm for N-Queens

There is no general linear-time algorithm to find *all* solutions of the N-Queens problem. However, a valid solution can be *constructed* in $\mathcal{O}(n)$ time for all $n \geq 4$.

Represent the solution by an array $Q[1 \dots n]$, where $Q[i]$ denotes the column position of the queen in row i.

Algorithm:

- If $n = 1$, the solution is [1].
- If $n = 2$ or $n = 3$, no solution exists.
- If n is even, place queens in columns:

$$2, 4, 6, \dots, n, 1, 3, 5, \dots, n-1$$

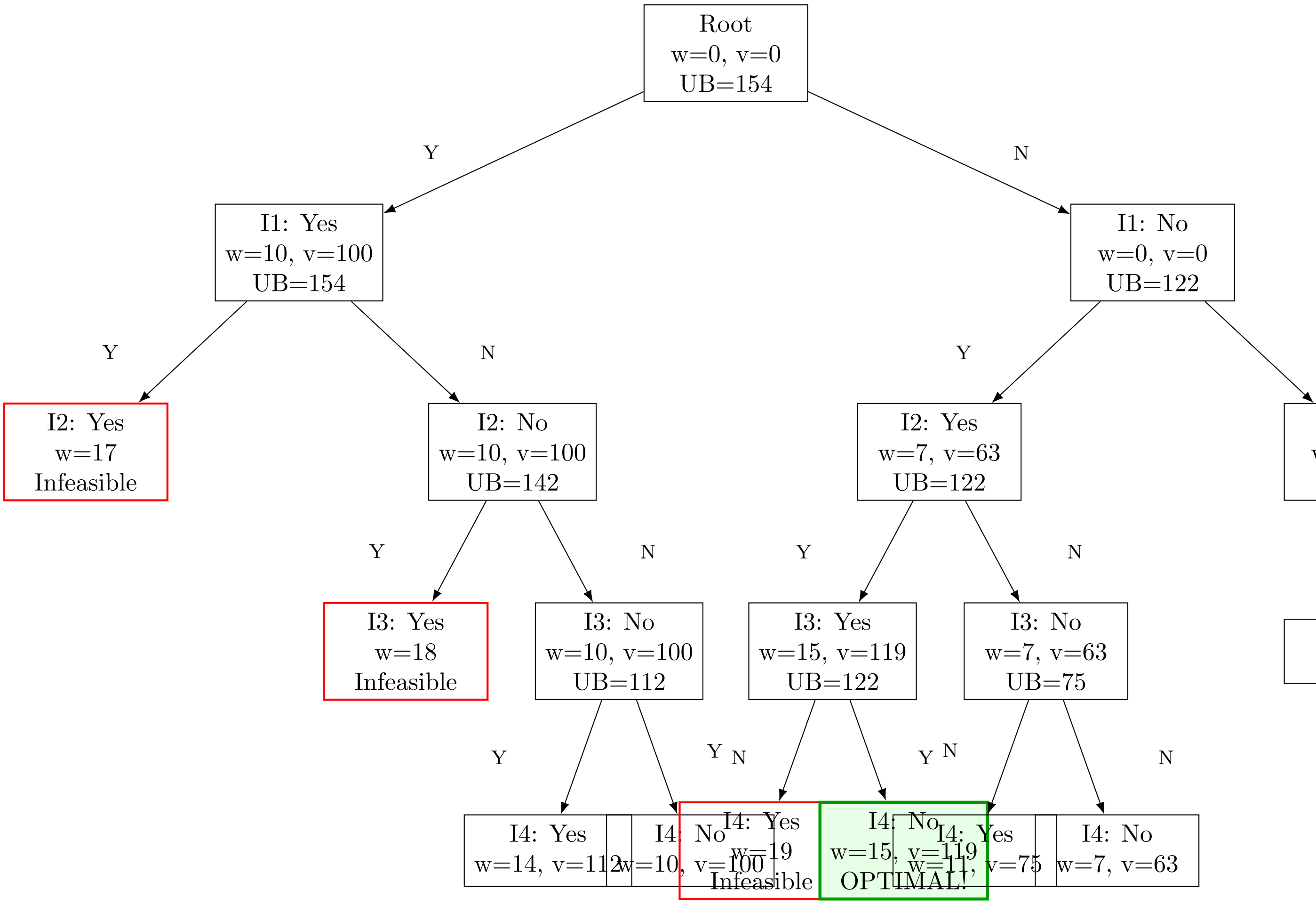
- If n is odd ($n \geq 5$), apply the even case for $n-1$ and place the last queen at (n, n).

This construction avoids column and diagonal conflicts and runs in $\mathcal{O}(n)$ time with $\mathcal{O}(n)$ space.

16 Qs 3

Branch and Bound Tree for Knapsack Problem

Problem: W = 16
Item 1: w=10, v=100, ratio=10.0
Item 2: w=7, v=63, ratio=9.0
Item 3: w=8, v=56, ratio=7.0
Item 4: w=4, v=12, ratio=3.0



Key Upper Bound Calculations:

- **Root:** $0 + 100 + (6/7)\times 63 = 154$
- **I1=Yes, I2=No:** $100 + (6/8)\times 56 = 100 + 42 = 142$
- **I1=Yes, I2=No, I3=No:** $100 + 12 = 112$
- **I1=No:** $0 + 63 + 56 + (1/4)\times 12 = 122$
- **I1=No, I2=Yes:** $63 + 56 + (1/4)\times 12 = 122$
- **I1=No, I2=Yes, I3=Yes:** $119 + (1/4)\times 12 = 122$

Optimal Solution: Select **Items 2 and 3**
Total Weight: $7 + 8 = 15 \leq 16$
Total Value: $63 + 56 = \mathbf{119}$

Note: The greedy approach (highest ratio first) gives Items 1+4 with value 112, but Branch and Bound explores all branches and finds the true optimal: Items 2+3 with value 119.

16.1 Qs 4

Step	i	j
1	0	0
2	1	1
3	2	2
4	2	0
5	3	1
6	3	0
7	4	1
8	4	0
9	5	1
10	6	2
11	7	3
12	8	4
13	9	5
14	10	6
15	10	2
16	11	3
17	11	0
18	12	0
19	13	1
20	14	2
21	14	0
22	15	1
23	16	2
24	16	0
25	17	1

Table 1: KMP Algorithm trace for text ”abaaabbaabbbababa” and pattern ”abbaab”