

Algorithms

(Backtracking)

Pramod Ganapathi

Department of Computer Science
State University of New York at Stony Brook

November 30, 2021



Contents

-  **Code**
-  Permutations
-  Subsets
-  Compositions
-  Diophantine
-  Parenthesizations
-  n -Queens
-  Derangements
-  Palindromes

Contributors

- Akansha Maloo

Backtracking framework

BACKTRACK(*depth*)

1. **if** REJECT(*depth*) **then**
2. **return**
3. **else if** ACCEPT(*depth*) **then**
4. OUTPUT(*depth*)
5. **else**
6. **for** *i* : LOOP(*depth*) **do**
7. **if** CONSTRAINT(*depth*, *i*) **then**
8. ALLOCATERESOURCES(*depth*, *i*)
9. BACKTRACK(*depth* + 1)
10. DEALLOCATERESOURCES(*depth*, *i*)

Backtracking framework

BACKTRACK(*depth*)

1. **if** REJECT(*depth*) **then**
2. **return**
3. **else if** ACCEPT(*depth*) **then**
4. OUTPUT(*depth*)
5. **else**
6. **for** *i* : LOOP(*depth*) **do**
7. **if** CONSTRAINT(*depth*, *i*) **then**
8. ALLOCATERESOURCES(*depth*, *i*)
9. BACKTRACK(*depth* + 1)
10. DEALLOCATERESOURCES(*depth*, *i*)

Advantages over existing backtracking frameworks

- Flexible pruning
- Generic types
- Modularity

Permutations

[HOME](#)

Step 1. Problem

- Generate r -permutations of u unique elements $item[1..u]$ using the $count[1..u]$ array. Here $count[i]$ presents the number of occurrences of element $item[i]$ for $i \in [1, u]$. We can assume that total number of items is $n = \sum_{i=1}^u count[i]$.
- For example, let $r = 3$, $u = 2$, $item = [a, b]$, and $count = [2, 2]$. Then the 3-permutations of $[a, a, b, b]$ is $\{[a, a, b], [a, b, a], [a, b, b], [b, a, a], [b, a, b], [b, b, a]\}$.

Step 2. Algorithm

PERMUTATIONS(*depth*)

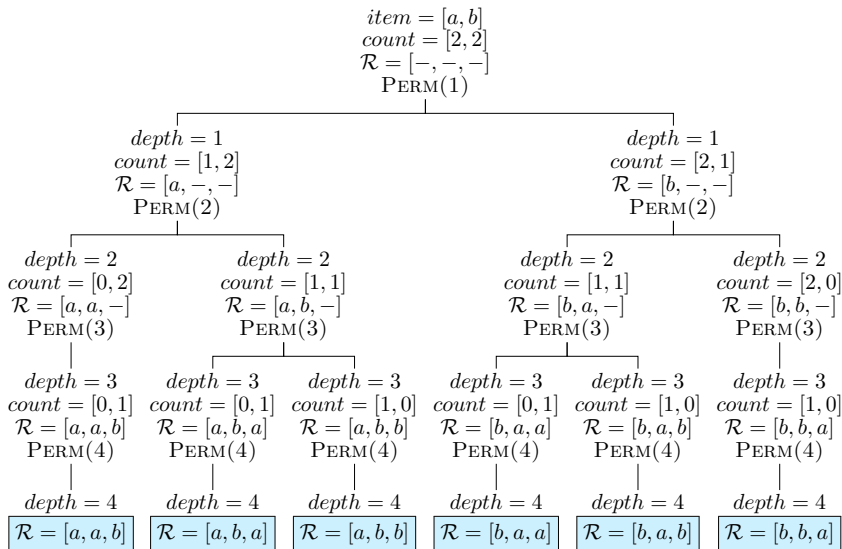
Input: Local: *depth*; Global: permutation size r , combinatorial object $\mathcal{R}[1..r]$
#unique items u , unique elements $item[1..u]$, #occurrences $count[1..u]$

Output: Generate all r -sized permutations of unique elements $item[1..u]$
having #occurrences $count[1..u]$

Require: Invoke PERMUTATIONS(1)

1. **if** $depth > r$ **then print** $\mathcal{R}[1..(depth - 1)]$
2. **else**
3. **for** $i \leftarrow 1$ **to** u **do**
4. **if** $count[i] \geq 1$ **then**
5. $\mathcal{R}[depth] \leftarrow item[i]$
6. $count[i] --$
7. PERMUTATIONS($depth + 1$)
8. $count[i] ++$

Step 3. Example



Subsets

HOME

Step 1. Problem

- Generate all subsets of u unique elements $item[1..u]$ using the $count[1..u]$ array. Here $count[i]$ presents the number of occurrences of element $item[i]$ for $i \in [1, u]$. We can assume that total number of items is $n = \sum_{i=1}^u count[i]$.
- For example: let $u = 3, item = [a, b, c], count = [1, 2, 1]$. Then the subsets of $[a, b, b, c]$ are $\{[], [a], [b], [c], [a, b], [a, c], [b, b], [b, c], [a, b, b], [a, b, c], [b, b, c], [a, b, b, c]\}$.

Step 2. Algorithm

SUBSETS(*depth*)

Input: Local: *depth*, Global: combinatorial object \mathcal{R} , #unique elements u , unique elements *item*[1.. u], #occurrences *count*[1.. u]

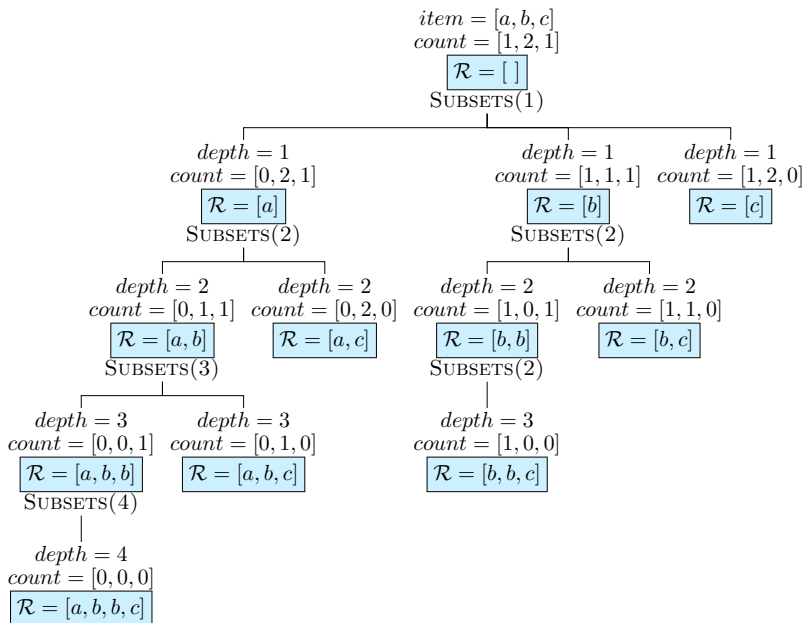
Output: Generate all subsets of unique elements *item*[1.. u] having #occurrences *count*[1.. u]

Require: Invoke SUBSETS(1)

1. **print** $\mathcal{R}[1..(\text{depth} - 1)]$
2. **if** $\text{depth} > n$ **then return**
3. **else**
4. **for** $i \leftarrow 1$ **to** u **do**
5. **if** $\text{count}[i] \geq 1$ **and**
 $(\text{depth} = 1 \text{ or } \text{item}[i] \geq \mathcal{R}[\text{depth} - 1])$ **then**
6. $\mathcal{R}[\text{depth}] \leftarrow \text{item}[i]$
7. $\text{count}[i] - -$
8. SUBSETS($\text{depth} + 1$)
9. $\text{count}[i] + +$

- $\text{depth} = 1$ means that at depth 1 the loop runs for u times.
- $\text{item}[i] \geq \mathcal{R}[\text{depth} - 1]$ means that the next element must always be greater than or equal to the latest element $\mathcal{R}[\text{depth} - 1]$.

Step 3. Example



Compositions

HOME

Step 1. Problem

- Generate all compositions of a given natural number n using u unique elements $item[1..u]$ using the $count[1..u]$ array. Here $count[i]$ presents the number of occurrences of element $item[i]$ for $i \in [1, u]$.
- Compositions are all arrangements such that the sum of elements equals n .
- For example: let $n = 4, u = 4, item = [1, 2, 3, 4], count = [\infty, \infty, \infty, \infty]$.
Compositions of 4 are $\{[1, 1, 1, 1], [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2], [1, 3], [3, 1], [4]\}$.

Step 2. Algorithm

COMPOSITIONS(*depth*)

Input: Local: *depth*, Global: combinatorial object \mathcal{R} , #unique elements *u*, unique elements *item*[1..*u*], #occurrences *count*[1..*u*]

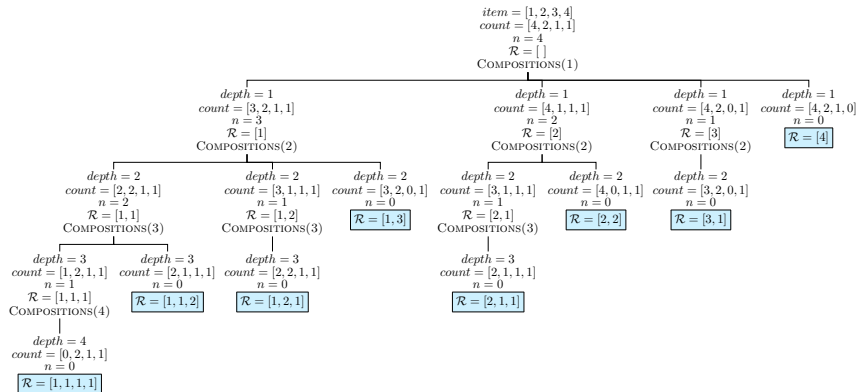
Output: Generate all compositions of *n* using unique elements *item*[1..*u*] having #occurrences *count*[1..*u*]

Require: Invoke COMPOSITIONS(1)

1. **if** $n = 0$ **then print** $\mathcal{R}[1..(depth - 1)]$
2. **else**
3. **for** $i \leftarrow 1$ **to** *u* **do**
4. **if** $count[i] \geq 1$ **and** $n \geq item[i]$ **then**
5. $\mathcal{R}[depth] \leftarrow item[i]$
6. $count[i] - -; n \leftarrow n - item[i]$
7. COMPOSITIONS(*depth* + 1)
8. $count[i] + +; n \leftarrow n + item[i]$

- $n \geq item[i]$ means that it is possible to add *item*[*i*] without making *n* negative.

Step 3. Example



Diophantine [HOME](#)

Step 1. Problem

- Generate all nonnegative solutions of a diophantine equation with u variables having positive integer coefficients $item[1..u]$ as follows:

$$\sum_{i=1}^u (item[i] \times x_i) = n$$

- $count[i] \leq n/item[i]$.
- For example: Find noninteger solutions to $2x_1 + 3x_2 = 24$.
That is $u = 2, item = [2, 3], count = [24/2, 24/3] = [12, 8]$.
Solutions are: $\{[0, 8], [3, 6], [6, 4], [9, 2], [12, 0]\}$.

Step 2. Algorithm

DIOPHANTINE($depth$)

Input: Local: $depth$, Global: combinatorial object \mathcal{R} , value n , #variables u , positive integer coefficients $item[1..u]$, #occurrences $count[1..u]$, where $count[i] \leq n/item[i]$

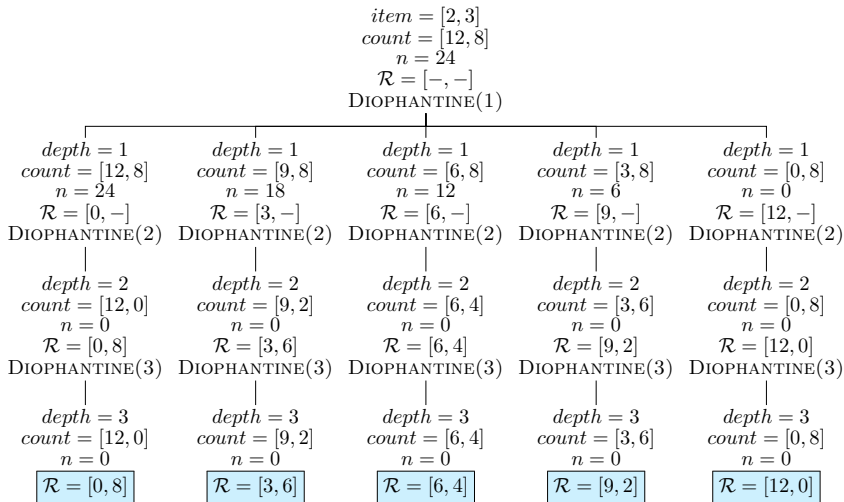
Output: Generate all nonnegative integer solutions to Diophantine equation.

Require: Invoke DIOPHANTINE(1)

1. **if** $depth = (u + 1)$ **and** $n \neq 0$ **then return**
2. **else if** $depth = (u + 1)$ **and** $n = 0$ **then print** $\mathcal{R}[1..(depth - 1)]$
3. **else**
4. **for** $i \leftarrow 0$ **to** $count[depth]$ **do**
5. **if** $n \geq i \times item[depth]$ **then**
6. $\mathcal{R}[depth] \leftarrow i$
7. $n \leftarrow n - i \times item[depth]$
8. DIOPHANTINE($depth + 1$)
9. $n \leftarrow n + i \times item[depth]$

- $n \geq i \times item[depth]$ means that it is possible to add i instances of $item[depth]$ without making n negative.

Step 3. Example



Parenthesizations

[HOME](#)

Step 1. Problem

- Generate valid parenthesizations with n open and n close parentheses.
- For example: when $n = 3$, valid parenthesizations are:
 $\{ [((()))], [((), ())], [(())()], [()(())], [()()()] \}.$

Step 2. Algorithm

PARENTHESIS($depth$)

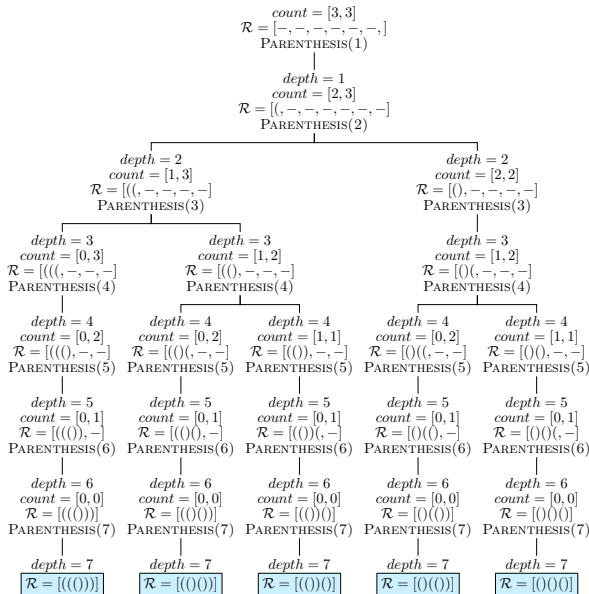
Input: Local: $depth$, Global: $item[1]$ - '(', $item[2]$ - ')', $count[1] = count[2] = n$, valid parenthesizations $\mathcal{R}[1..2n]$

Output: Generate all valid parenthesizations with n open and n close brackets.

1. **if** $depth > 2n$ **then print** $\mathcal{R}[1..(depth - 1)]$
2. **else**
3. **for** $i \leftarrow 1$ **to** 2 **do**
4. **if** $count[i] \geq 1$ **and** ($i = 1$ **or** $count[2] > count[1]$) **then**
5. $\mathcal{R}[depth] \leftarrow item[i]$
6. $count[i] --$
7. PARENTHESIS($depth + 1$)
8. $count[i] ++$

- $i = 1$ means that any number of left braces can be added
- $count[2] > count[1]$ means that the available instances of $item[2]$ is more than the available instances of $item[1]$

Step 3. Example



n-Queens [HOME](#)

Step 1. Problem

- Place n queens on an $n \times n$ chessboard such that no two queens fight (i.e., no two queens must be in the same row or same column or same left diagonal or same right diagonal).
- For example: when $n = 4$, there are two solutions:
 $\{[2, 4, 1, 3], [3, 1, 4, 2]\}$.

Step 2. Algorithm

NQUEENS($depth$)

Input: Local: $depth$, Global: $col[1..n] \leftarrow \{1\}$, $leftdiag[1..2n-1] \leftarrow \{1\}$, $rightdiag[1..2n-1] \leftarrow \{1\}$, n queens solutions $\mathcal{R}[1..n]$

Output: Generate all solutions to n queens problem

1. **if** $depth > n$ **then print** $\mathcal{R}[1..(depth-1)]$
2. **else**
3. **for** $i \leftarrow 1$ **to** n **do**
4. **if** $col[i] = 1$
 and $leftdiag[n+depth-i] = 1$
 and $rightdiag[depth+i-1] = 1$ **then**
5. $\mathcal{R}[depth] \leftarrow i$
6. $col[i] --$; $leftdiag[n+depth-i] --$; $rightdiag[depth+i-1] --$
7. **NQUEENS**($depth+1$)
8. $col[i] ++$; $leftdiag[n+depth-i] ++$; $rightdiag[depth+i-1] ++$

Derangements

HOME

Step 1. Problem

- Generate all r -sized derangements for a given set of elements. A derangement is a permutation such that no element appears in its original position.
- For example: let $r = 3$, $u = 3$, $item = [1, 2, 3]$, $count = [1, 2, 2]$. Then the derangements are:
 $\{[2, 1, 2], [2, 3, 1], [2, 3, 2], [3, 1, 2], [3, 3, 1], [3, 3, 2]\}$.

Step 2. Algorithm

DERANGEMENTS(*depth*)

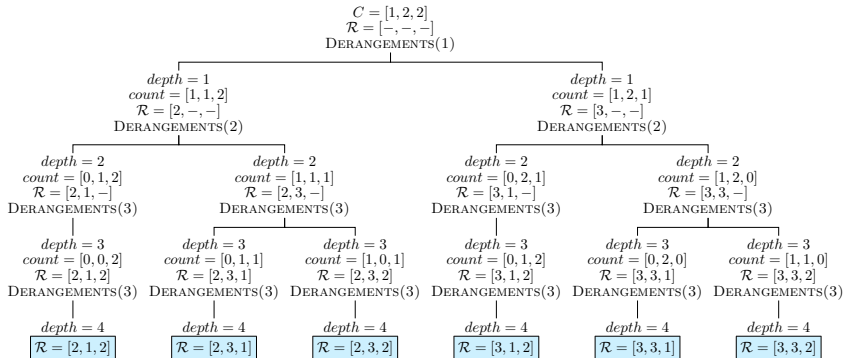
Input: Local: *depth*; Global: derangement size *r*, combinatorial object $\mathcal{R}[1..r]$, #unique items *u*, unique elements *item*[1..*u*], #occurrences *count*[1..*u*]

Output: Generate all *r*-sized derangements of unique elements *item*[1..*u*] having #occurrences *count*[1..*u*]

Require: Invoke DERANGEMENTS(1)

1. **if** *depth* > *r* **then print** $\mathcal{R}[1..(depth - 1)]$
2. **else**
3. **for** *i* \leftarrow 1 **to** *u* **do**
4. **if** *count*[*i*] \geq 1 **and** *item*[*i*] \neq *depth* **then**
5. $\mathcal{R}[depth] \leftarrow item[i]$
6. *count*[*i*] – –
7. DERANGEMENTS(*depth* + 1)
8. *count*[*i*] + +

Step 3. Example



Palindromes

[HOME](#)

Step 1. Problem

- Generate all r -palindromes of a given set of elements.
- For example: When $u = 3$, $r = 4$, $item = [a, b, c]$, $count = [2, 2, 2]$, the r -palindromes are $\{[a, b, b, a], [a, c, c, a], [b, a, a, b], [b, c, c, b], [c, a, a, c], [c, b, b, c]\}$.
When $u = 3$, $r = 3$, $item = [a, b, c]$, $count = [2, 2, 2]$, the r -palindromes are $\{[a, b, a], [a, c, a], [b, a, b], [b, c, b], [c, a, c], [c, b, c]\}$.

Step 2. Algorithm

PALINDROMES($depth$)

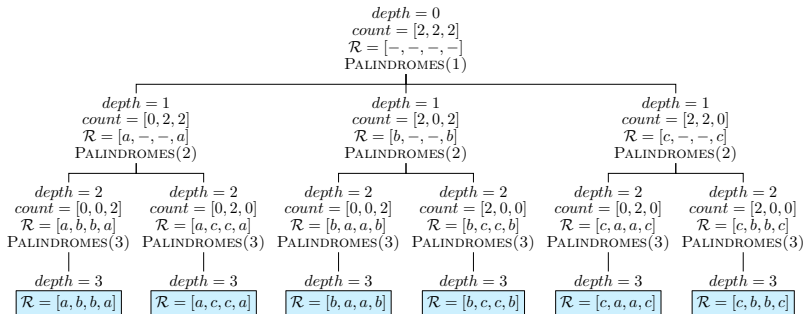
Input: Local: $depth$; Global: palindrome size r , combinatorial object $\mathcal{R}[1..r]$
#unique items u , unique elements $item[1..u]$, #occurrences $count[1..u]$

Output: Generate all r -sized palindromes of unique elements $item[1..u]$ having #occurrences $count[1..u]$

Require: Invoke PALINDROMES(1)

1. **if** $depth = \lfloor r/2 \rfloor + 1$ **then**
2. **if** r is even **then print** $\mathcal{R}[1..r]$
3. **else if** r is odd **then**
4. **for** $i = 1$ **to** u **do**
5. **if** $count[i] \geq 1$ **then**
6. $\mathcal{R}[depth] \leftarrow item[i]$
7. **print** $\mathcal{R}[1..r]$
8. **else**
9. **for** $i \leftarrow 1$ **to** u **do**
10. **if** $count[i] \geq 2$ **then**
11. $\mathcal{R}[depth] \leftarrow \mathcal{R}[r - depth + 1] \leftarrow item[i]$
12. $count[i] \leftarrow count[i] - 2$
13. PALINDROMES($depth + 1$)
14. $count[i] \leftarrow count[i] + 2$

Step 3. Example (r is even)



Step 3. Example (r is odd)

