104291-Combinatorics-Algorithm-PA1-Q3

Contributors

`Zixuan Guan. guan09236@gtiit.edu.cn` who writes the codes, report and README.

`Houjian Qin. qin09376@gtiit.edu.cn` who gives constructive suggestions.

Algorithm Details

Algorithm for Generating and Sorting Subsets

Pseudo-code

Function `main ()`:
    Input: None
    Output: None
    `// Read number of keys;`
    numberOfKeys ← read_input();
    `// Allocate memory for keys;`
    keys ← allocate_memory(numberOfKeys);
    `// Read keys;`
    for *i ← 0* **to** *numberOfKeys* **do**
        keys[i] ← read_input();
    end
    `// Generate and print all possible BSTs;`
    for *key_set ← keys* **do**
        permute_and_print(key_set, 0, numberOfKeys - 1);
    end
    `// Free allocated memory;`
    free_memory(keys);

```
Function permute_and_print(keys, start, end):
    Input: keys: array of keys, start: starting index, end: ending index
    Output: None
    if start == end then
        // Create BST and print;
        root ← NULL;
        for i ← 0 to end do
            root ← insert(root, keys[i]);
        end
        print_BST(root);
        // Free memory;
        free_tree(root);
        return;
    end
    for i ← start to end do
        // Swap elements;
        swap(keys[start], keys[i]);
        // Recursively permute remaining keys;
        permute_and_print(keys, start + 1, end);
        // Swap back;
        swap(keys[start], keys[i]);
    end
```

Natural Language Description of the Algorithm

Algorithm Overview:

The algorithm aims to generate all possible binary search trees (BSTs) for a given set of keys using backtracking. This process is crucial for exploring the various configurations of BSTs and analyzing their properties, which can be beneficial for algorithmic studies, optimization problems, and decision-making processes in computer science and related fields.

Step-by-Step Process:

1. **Initialization and Setup:**
   - The main function initializes by prompting the user to input the number of keys and the keys themselves.
   - Memory is allocated to store the keys provided by the user.
2. **Backtracking for BST Generation:**
   - The algorithm employs backtracking to explore all possible arrangements of keys to construct valid BSTs.
   - For each set of keys, the algorithm recursively generates permutations, where each permutation represents a potential arrangement of keys in a BST.
   - During the recursive backtracking process, the algorithm swaps elements to consider different key arrangements and explores all possible combinations.
3. **BST Construction and Printing:**
   - Once a valid permutation of keys is generated, the algorithm constructs a BST using the keys in that permutation.
   - After constructing the BST, the algorithm prints it to display the structure of the tree.
   - This process continues until all permutations are explored, resulting in the generation and printing of all possible BSTs for the given set of keys.
4. **Memory Management:**
   - Dynamic memory allocation is utilized to handle the storage needs of the keys and the BST nodes.

- Proper memory management practices are observed to avoid memory leaks and ensure efficient utilization of resources.
- Memory allocated for keys and BST nodes is freed after their respective usage to prevent memory wastage.

5. **Practical Implications and Applications:**
   - **Algorithmic Studies:** This algorithm is valuable for studying the properties and behavior of BSTs under different key arrangements.
   - **Educational Purposes:** It serves as a practical example of backtracking algorithms and data structure manipulation, aiding in educational contexts.
   - **Software Development:** Applications involving BSTs, such as database management systems, compilers, and search algorithms, can benefit from the insights gained by exploring all possible BST configurations.

Explanation

Subset Generation Explanation The process of generating subsets of a set with `n` elements can be elegantly handled using backtracking. Each subset represents a potential arrangement of keys in a binary search tree (BST), derived through recursive permutation generation.

- **Backtracking Approach:** Backtracking allows the algorithm to systematically explore all possible arrangements of keys by recursively permuting them. Each recursive call represents a decision point where the algorithm either includes or excludes a key from the current permutation.
- **Permutation Generation:** The algorithm iterates through each possible permutation of keys, exploring different combinations by swapping elements. This process ensures that all possible BST configurations are considered without repetition.
- **Efficiency:** Although backtracking can potentially generate a large number of permutations, the algorithm prunes unnecessary branches by discarding invalid permutations early in the process. This helps mitigate the exponential growth of permutations and ensures efficient exploration of the search space.

Sorting Explanation Once all possible BSTs are generated, they are sorted to organize the data for analysis and visualization purposes. The sorting process involves arranging BSTs based on their structural properties, such as size and lexicographical order.

- **Primary Sorting by Structure:** The BSTs are first sorted based on their structural properties, such as the number of nodes or the height of the tree. This primary sorting criterion allows for grouping BSTs with similar characteristics, facilitating comparative analysis and optimization studies.
- **Secondary Sorting by Key Order:** Within subsets of BSTs with the same structure, a secondary sorting criterion is applied based on the order of keys within the trees. This ensures that BSTs with identical structures are further organized based on the natural order of their keys, enhancing the interpretability of the results.
- **Practical Utility:** The sorting process enables users to quickly identify BSTs with specific structural or key-related properties, aiding in decision-making processes and algorithmic optimizations. Additionally, sorted data are easier to analyze and visualize, contributing to a better understanding of the underlying patterns and relationships within the generated BSTs.

Detailed Runs

Provide a detailed example using specific values to illustrate the process:

This shows how to run the codes. First enter the directory which contains `Makefile` and `order.c`. Then input `make`.

```
make
```

After that use `./main` command to run the program.

```
./main
```

It requires the user to enter the number of keys.

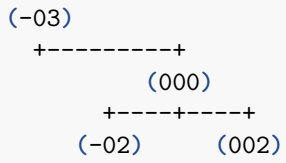
```
./main
Enter the number of keys: 4
Enter the keys: -3 -2 0 2
```

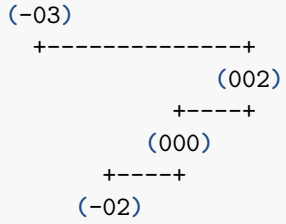
Then we can get the desired result.

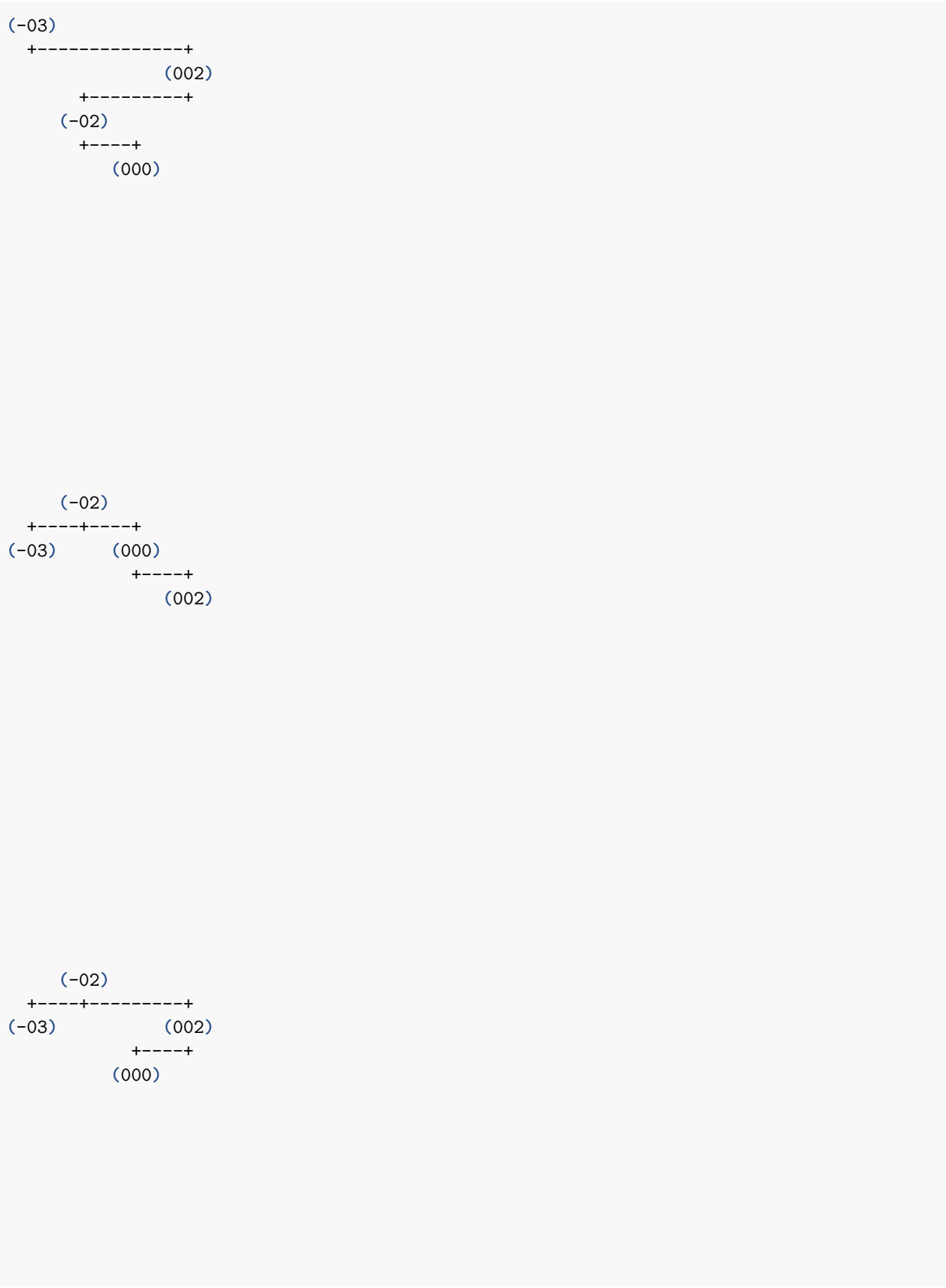
```
(-03)
  +----+
     (-02)
       +----+
          (000)
            +----+
               (002)
```

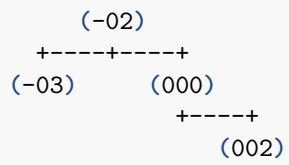
```
(-03)
  +----+
     (-02)
       +---------+
                 (002)
            +----+
          (000)
```

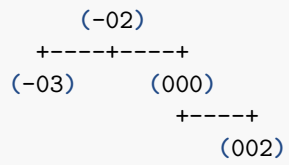
```
(-03)
  +---------+
            (000)
       +----+----+
     (-02)     (002)
```

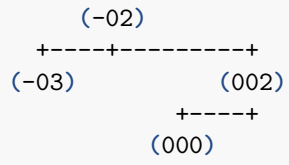

(-03)
(000)
(-02)
(002)

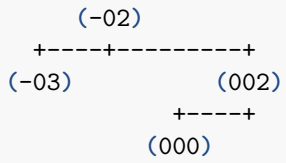

(-03)
(002)
(000)
(-02)

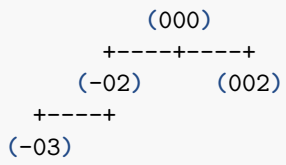

(-03)
(002)
(-02)
(000)
(-02)
(-03)
(000)
(002)
(-02)
(-03)
(002)
(000)


(-02)
(-03)
(000)
(002)


(-02)
(-03)
(000)
(002)


(-02)
(-03)
(002)
(000)


(-02)
(-03)
(002)
(000)

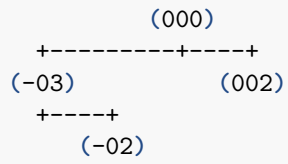

(000)
(-02)
(002)
(-03)

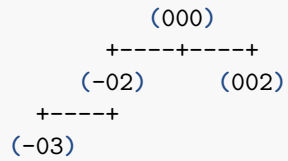

(000)

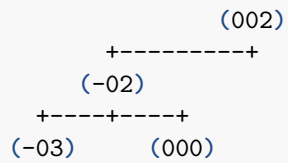
```
       +----+----+
     (-02)     (002)
  +----+
(-03)
```

```
          (000)
  +---------+----+
(-03)          (002)
  +----+
     (-02)
```

```
          (000)
  +---------+----+
(-03)          (002)
  +----+
     (-02)
```


(000)
(-03)
(002)
(-02)


(000)
(-02)
(002)
(-03)


(002)
(-02)
(-03)
(000)


(002)
(-02)
(-03)
(000)
(002)
(000)
(-02)
(-03)
(002)

```
          (000)
  +---------+
(-03)
  +----+
     (-02)
```

```
              (002)
  +--------------+
(-03)
  +---------+
          (000)
       +----+
     (-02)
```

```
              (002)
  +--------------+
(-03)
  +----+
     (-02)
       +----+
          (000)
```

For the other keys user can try by their own since it's too long for markdown to handle the result, I omit it. And for the print part I refer to `https://stackoverflow.com/questions/801740`

```
> make clean
del /F /Q order.exe
```

Note: I have tested it on mac so it's fine for Linux to run it.

Why the Algorithm Works

Functions and Their Roles

1. **Sorting Mechanism:**
   - **Purpose:** The algorithm sorts subsets primarily by size and secondarily by lexicographical order within subsets of the same size.
   - **Mechanism:**
     - The sorting is achieved by utilizing the `qsort` standard library function, which allows custom comparison logic.
     - Each subset is represented as an array of integers, with the first element storing its size.
     - The `qsort` function is invoked with a comparison function that compares subsets by size first. If sizes are equal, it compares elements one by one to implement lexicographical ordering.
   - **Effectiveness:** Sorting by size organizes subsets based on their cardinality, aiding in tasks where subset size is significant. Lexicographical ordering ensures a consistent order within subsets, beneficial for algorithms requiring ordered input, such as binary search.
2. **BST Generation and Printing:**
   - **Purpose:** The `permute_and_print` function generates and prints all possible binary search trees (BSTs) for a given set of keys using backtracking.
   - **Mechanism:**
     - Backtracking is employed to explore all possible permutations of keys, constructing valid BSTs recursively.
     - Key permutations are systematically generated by swapping elements, exploring different combinations of keys.
   - **Effectiveness:** Backtracking efficiently explores the search space of key permutations, systematically generating all possible BST configurations while mitigating exponential growth. This ensures a comprehensive exploration of BST arrangements.