Subset Sum Problem

Team Name: A Parallel Team

Team Members:
Alimuddin Khan
Karan Jariwala
Agenda

- Overview of the Subset sum problem
- Design and operation of sequential program
- Design and operation of parallel program
- Demo of the code
1. Overview of Subset Sum Problem
What is a Subset Sum Problem (SSP)?
What is a Subset Sum Problem?

▸ Given a set of positive integers \( N \) and a sum \( S \), does there exist a subset \( N' \subseteq N \) such that sum of all the elements in that subset is \( S \).

▸ Mathematically,
\[
S = \sum_{n \in N'} n
\]
\( N' \in \text{PowerSet}(N) \)

▸ It is a NP-Complete problem
▸ It is special case of 0-1 KnapSack problem.
Example of Subset Sum Problem

- Consider set $N = \{7, 5, 1, 9, 6, 2\}$ and $S = 3$
- Then by observing we can say that $N’ = \{1, 2\}$
- This is easy to find when number of elements in a set is small.
Naïve Approach
Input set $W = \{3, 7, 5, 10, 11, 1\}$ of size $N$ (where $N = 6$ here)

Generate all possible subsets as set of set data structure with naïve approach. E.g.: For $N$ elements, it will generate $2^N$ possible subsets

Iterate through all the subsets and their sum compare with the target sum. If matches then return True else repeat until all subsets are covered.

If no match found, then return False

Possible Subsets:

$\{
\}
\{1\}
\{3\}
\{1 3\}
\{5\}
\{1 5\}
\{7\}
\{1 7\}
\{3 5\}
\{1 3 5\}
\ldots$

Target Sum: 4

Output: True
Design and implementation of sequential program
BitSet64: Asymptotic space complexity is reduced from $O(2^N)$ to $O(N^2)$

Let's say if we want to add the subset sum \{1, 3, 61, 66\} to the ArrayList<BitSet64>
Sequential Generation and Merging code:

```java
// initialize first element
subSets.get(0).add(0);

// subset sum generation stage
for(int element: list){
    // taking one element at a time and adding it to all the subset sums
    // calculated in previous step
    for(int i = 0; i < bitSetsRequired*64; i++){
        if(subSets.get(i/64).contains(i%64)){
            temp.get((element + i)/64).add((element + i)%64);
        }
    }

    // merging section
    for(int i = 0; i < subSets.size(); i++){
        // take union to merge both the subset sums
        subSets.get(i).union(temp.get(i));

        // clear the temp subset
        temp.get(i).clear();
    }
}

return subSets;
```

Input set \(W = \{3, 7, 5, 10, 11, 1\}\) of size \(N\) (where \(N = 6\) here)

First half list \(W_1 = \{3, 7, 5\}\) of size \(N/2\), then Second half list \(W_2 = \{10, 11, 1\}\) of size \(N/2\)

Create a SubsetSum1 from \(W_1\), then SubsetSum2 from \(W_2\) of types ArrayList\(<\)BitSet64\(>\)

Possible subsets from \(W_1\) are \(\{0, 3, 7, 10, 5, 8, 12, 15\}\) and \(W_2\) are \(\{0, 1, 10, 11, 11, 12, 21, 22\}\)

Add the first element from both the list, if the sum is smaller than the targetSum then increment the list1 index, else if sum is larger than the targetSum then decrement the list2 index, and if sum is equal to the targetSum then return True. Repeat this process until it founds the answer or till the end of the list.
Input Set $W = \{3, 7, 5, 10, 11, 1\}$
List1 $W_1 = \{3, 7, 5\}$ and List2 $W_2 = \{10, 11, 1\}$

Let’s understand first the steps for creating SubsetSum1 from $W_1$:

Step 1: SubsetSum1 = \{0\}
W1[0] = \{3\}
tempList = W1[0] + SubsetSum1 = \{3\}
SubsetSum1 = SubsetSum1 U tempList = \{0, 3\}

Step 2: SubsetSum1 = \{0, 3\}
W1[1] = \{7\}
tempList = W1[1] + SubsetSum1 = \{7, 10\}
SubsetSum1 = SubsetSum1 U tempList = \{0, 3, 7, 10\}

Step 3: SubsetSum1 = \{0, 3, 7, 10\}
W1[2] = \{5\}
tempList = W1[2] + SubsetSum1 = \{5, 8, 12, 15\}
SubsetSum1 = SubsetSum1 U tempList = \{0, 3, 5, 7, 8, 10, 12, 15\}

Similar steps for creating SubsetSum2 from $W_2$:
public static boolean isPresent(ArrayList<BitSet64> list1,
        ArrayList<BitSet64> list2,
        int targetSum)
{
    int i = 0;
    int j = list2.size()*64 - 1;

    while(i < list1.size()*64 && j >= 0){
        // skip all the bits which are not set
        while(i/64 < list1.size() && !list1.get(i/64).contains(i%64)){
            i++;
        }

        // skip all the bits which are not set
        while(j/64 >= 0 && !list2.get(j/64).contains(j%64)){
            j--;
        }

        // making sure if we are still in the range
        if(i >= list1.size()*64 || j < 0 ){
            return false;
        }

        if(targetSum == i || targetSum == j || targetSum == (i+j)){
            return true;
        }else if(targetSum > (i + j)){
            // moving forward in list 1
            i++;
        }else{
            // moving backward in list 2
            j--;
        }
    }

    return false;
}
Move forward if target sum is greater than the sum of elements currently pointed by arrows.

Move backward if target sum is smaller than the sum of elements currently pointed by arrows.
Design and implementation of parallel program
Parallel Approach:

// calculating subset sums of both the lists in parallel
parallelDo(new Section()) {
    @Override
    public void run() throws Exception {

        // running a parallel for loop to add current element
        // to all current present elements in the subset sums bitSet ArrayList
        parallelFor(0, bitSetsRequired * 64 - 1).schedule(guided).exec(new Loop() {
            @Override
            public void run(int i) throws Exception {
                if (subsetSum1.get(i / 64).contains(i % 64)) {
                    temp1.get((currentElement1.item + i) / 64).add((currentElement1.item + i) % 64);
                }
            }
        });

        // parallelly merging and resetting the temp set
        parallelFor(0, subsetSum1.size() - 1).exec(new Loop() {
            @Override
            public void run(int i) throws Exception {
                // take union to merge both the subset sums
                subsetSum1.get(i).union(temp1.get(i));
                // clear the temp subset
                temp1.get(i).clear();
            }
        });
    }
}

First half list W1 = {3, 7, 5} of size N/2
Second half list W2 = {10, 11, 1} of size N/2

Create a SubsetSum1 of type ArrayList<BitSet64>
Create a SubsetSum2 of type ArrayList<BitSet64>

For all K threads in parallel:
Produce a new list A’ by adding the item Wi+1 to each element of the list Ai
e.g: {0}, {0,3}, {0,3,7,10} etc

Optimal parallel merging algorithm to merge the two list A and A’ to get the final arraylist as {0, 3, 5, 7, 8, 10, 12, 15}

For all K threads in parallel:
Produce a new list A’ by adding the item Wi+1 to each element of the list Ai
e.g: {0}, {0,10}, {0,10,11,21} etc

Optimal parallel merging algorithm to merge the two list A and A’ to get the final arraylist as {0, 1, 10, 11, 12, 21, 22}

In a searching phase, the main thread will perform the search operation as we discussed in sequential program
DEMO
References
References


THANKS!

Any questions?
You can reach us at aak5031@rit.edu & kkj1811@rit.edu