# Recitation 20: Subset Sum Variants

## Subset Sum Review

- Input: Set of $n$ positive integers $A[i]$
- Output: Is there subset $A' \subset A$ such that $\sum_{a \in A'} a = S$?
- Can solve with dynamic programming in $O(nS)$ time
- (See lecture notes 19 for specifics)

## Subset Sum

- Input: Set of $n$ positive integers $A[i]$
- Output: Is there subset $A' \subset A$ such that $\sum_{a \in A'} a = S$?

1. **Subproblems**
   - Idea: Is last item in a valid subset? (Guess!)
   - If yes, then try to sum to $S - A[n] \geq 0$ using remaining items
   - If no, then try to sum to $S$ using remaining items
   - $x(i, j)$: $T$ if can make sum $j$ using items 1 to $i$, $F$ otherwise
2. **Relate**
   - $x(i, j) = \text{OR}\left\{ \begin{array}{ll} x(i-1, j-A[i]) & \text{if } j \geq A[i] \\ x(i-1, j) & \text{always} \end{array} \right\} \text{for } i \in [0, n], j \in [0, S]$
   - Subproblems $x(i, j)$ only depend on strictly smaller $i$, so acyclic
3. **Base**
   - $x(i, 0) = T$ for $i \in [0, n]$, $x(0, j) = F$ for $j \in [1, S]$
4. **Solution**
   - Solve subproblems via recursive top down or iterative bottom up
   - Maximum evaluated expression is given by $x(n, S)$
5. **Time**
   - (# subproblems: $O(nS)$) $\times$ (work per subproblem $O(1)$) $= O(nS)$ running time.

**Exercise:** Partition - Given a set of $n$ positive integers $A$, describe an algorithm to determine whether $A$ can be partitioned into two non-intersecting subsets $A_1$ and $A_2$ of equal sum, i.e. $A_1 \cap A_2 = \emptyset$ and $A_1 \cup A_2 = A$ such that $\sum_{a \in A_1} a = \sum_{a \in A_2} a$.
Example: $A = \{1, 4, 3, 12, 19, 21, 22\}$ has partition $A_1 = \{1, 19, 21\}$, $A_2 = \{3, 4, 12, 22\}$.

**Solution:** Run subset sum dynamic program with same $A$ and $S = \frac{1}{2} \sum_{a \in A} a$.

**Exercise:** Close Partition - Given a set of $n$ positive integers $A$, describe an algorithm to find a partition of $A$ into two non-intersecting subsets $A_1$ and $A_2$ such that the difference between their respective sums are minimized.

**Solution:** Run subset sum dynamic program as above, but evaluate for every $S' \in [0, \frac{1}{2} \sum_{a \in A} a]$, and return the largest $S'$ such that the subset sum dynamic program returns true.

**Exercise:** Can you adapt subset sum to work with negative integers?

**Solution:** Same as subset sum (see L19), but we allow calling subproblems with larger $j$. But now instead of solving $x(i, j)$ only in the range $i \in [0, n], j \in [0, S]$ as in positive subset sum, we allow $j$ to range from $j_{min} = -\sum_{a \in A, a<0} a$ (smallest possible $j$) to $j_{max} = \sum_{a \in A, a>0} a$ (largest possible $j$).

$$x(i, j) = \text{OR} \begin{cases} x(i-1, j-A[i]) & \text{if } j_{min} \leq j - A[i] \leq j_{max} \\ x(i-1, j) & \text{always} \end{cases}$$

Subproblem dependencies are still acyclic because $x(i, j)$ only depend on strictly smaller $i$. Base cases are $x(0, 0) = T$ and $x(0, j) = F$ if $j \neq 0$. Running time is then proportional to number of constant work subproblems, $O(n(j_{max} - j_{min}))$.

Alternatively, you can convert to an equivalent instance of positive subset sum and solve that. Choose large number $Q > \max(|S|, \sum_{a \in A} |a|)$. Add $2Q$ to each integer in $A$ to form $A'$, and append the value $2Q$, $n-1$ times to the end of $A'$. Now $A'$ are all positive, so solve positive subset sum with $S' = S + n(2Q)$. Because $(2n-1)Q < S' < (2n+1)Q$, any satisfying subset will contain exactly $n$ integers from $A'$ since sum of any fewer would have sum no greater than $(n-1)2Q + \sum_{a \in A} |a| < (2n-1)Q$, and sum of any more would have sum no smaller than $(n+1)2Q - \sum_{a \in A} |a| > (2n+1)Q$. Further, at least one integer in a satisfying subset of $A'$ corresponds to an integer of $A$. If $A'$ has a subset $B'$ summing to $S'$, then the items in $A$ corresponding to integers in $B'$ will comprise a nonempty subset that sum to $S$. Conversely, if $A$ has a subset $B$ that sums to $S$, choosing the $k$ elements of $A'$ corresponding the integers in $B$ and $n-k$ of the added $2Q$ values in $A'$ will comprise a subset $B'$ that sums to $S'$.

## 0-1 Knapsack

- Input: Knapsack with size $S$, want to fill with items each item $i$ has size $s_i$ and value $v_i$.
- Output: A subset of items (may take 0 or 1 of each) with $\sum s_i \leq S$ maximizing value $\sum v_i$
- (Subset sum same as 0-1 Knapsack when each $v_i = s_i$, deciding if total value $S$ achievable)
- Example: Items $\{(s_i, v_i)\} = \{(6,6),(9,9),(10,12)\}, S = 15$
- Solution: Subset with max value is all items except the last one (greedy fails)

1. **Subproblems**
    - Idea: Is last item in an optimal knapsack? (Guess!)
    - If yes, get value $v_i$ and pack remaining space $S - s_i$ using remaining items
    - If no, then try to sum to $S$ using remaining items
    - $x(i, j)$: maximum value by packing knapsack of size $j$ using items 1 to $i$
2. **Relate**
    - $x(i,j) = \begin{cases} \max(x(i-1,j), v_i + x(i-1, j-s_i)) & \text{if } j \geq s_i \\ x(i-1,j) & \text{otherwise} \end{cases}$
    - for $i \in [0, n], j \in [0, S]$
    - Subproblems $x(i, j)$ only depend on strictly smaller $i$, so acyclic
3. **Base**
    - $x(i, 0) = 0$ for $i \in [0, n]$, $x(0, j) = 0$ for $j \in [1, S]$
4. **Solution**
    - Solve subproblems via recursive top down or iterative bottom up
    - Maximum evaluated expression is given by $x(n, S)$
    - Store parent pointers to reconstruct items to put in knapsack
5. **Time**
    - # subproblems: $O(nS)$
    - work per subproblem $O(1)$
    - $O(nS)$ running time

**Exercise:** Close Partition (Alternative solution)

**Solution:** Given integers $A$, solve a 0-1 Knapsack instance with $s_i = v_i = A[i]$ and $S = \frac{1}{2}\sum_{a \in A} a$, where the subset returned will be one half of a closest partition.

**Exercise:** Unbounded Knapsack - Same problem as 0-1 Knapsack, except that you may take as many of any item as you like.

**Solution:**

1. **Subproblems:**
   - Idea: Guess an item in an optimal knapsack
   - If guess item $i$, receive value $v_i$, then pack remaining space $S - s_i$
   - $x(j)$: maximum value by packing knapsack of size $j$ using the provided items
2. **Relate:**
   - $x(j) = \max(0, \max\{v_i + x(j - s_i) \mid i \in [1, n],\ s_i \leq j\})$ for $j \in [0, S]$
   - Subproblems $x(j)$ only depend on strictly smaller $j$, so acyclic
3. **Base**
   - $x(0) = 0$ (no space to pack!)
4. **Solution**
   - Solve subproblems via recursive top down or iterative bottom up
   - Maximum evaluated expression is given by $x(S)$
   - Store parent pointers to reconstruct items to put in knapsack
5. **Time**
   - # subproblems: $O(S)$
   - work per subproblem $O(n)$
   - $O(nS)$ running time

We've made CoffeeScript visualizers solving both subset sum and 0-1 Knapsack, which you can find here:
`https://codepen.io/mit6006/pen/JeBvKe`
`https://codepen.io/mit6006/pen/VVEPod`