M1741 Fill in the Bag

Problem by Charlie Li

Problem statement

- Given N items with their weight w_i and the capacity of the bag K.
- Every kind of item is unlimited in supply.
- Find number of different ways to put those item into the bag.
- Two ways are different if their multiset is different or the order is different.
- ie. {1, 2} != {2, 1}
- Note: putting nothing into the bag ({}) is one of the way.
- Output the answer modulo 1000000007 ($10^9 + 7$).

Subtask

- For all cases: $1 \le N \le 500000$, $1 \le K \le 10^9$, $1 \le w_i \le 100$ for $1 \le i \le N$.
- Subtask 1 (7 points): N = 1
- Subtask 2 (24 points): All w_i are the same
- Subtask 3 (19 points): 1 <= N <= 5000 and 1 <= K <= 5000
- Subtask 4 (13 points): 1 <= K <= 5000
- Subtask 5 (37 points): No additional constraints

Subtask 1

- Subtask 1 (7 points): N = 1
- There is only one item, so we only need to consider how many to put into the bag.
- We can put 0, 1, 2, …, $\left\lfloor \frac{K}{w_1} \right\rfloor$ of item 1 into the bag.
- So there are totally $\left\lfloor \frac{K}{w_1} \right\rfloor + 1$ ways.

Subtask 2

- Subtask 2 (24 points): All w_i are the same
- Since all items are having the same weight, in addition to number of items, we need to consider the order also.
- We can put 0, 1, 2, …, $\left\lfloor \frac{K}{w} \right\rfloor$ items into the bag.
- If we are going to put i items into the bag, how many ways do we have?
  - Since order matters, so we have totally N^i ways.
- So the answer is

$$\sum_{i=0}^{\left\lfloor \frac{K}{w} \right\rfloor} N^i = 1 + N + N^2 + \cdots + N^{\left\lfloor \frac{K}{w} \right\rfloor}$$

Subtask 2

- To calculate the sum, we can use recursion
- Define f(x, y) = (x^0 + x^1 + … + x^y) % p where p = 10^9 + 7
- We can calculate f(x, y) recursively.
  - Base case: y = 0, f(x, y) = 1
  - If y%2 == 0 then f(x, y) = (f(x, y - 1) + x^y) % p
  - Otherwise f(x, y) = (f(x, (y-1)/2) * (1 + x^((y-1)/2 + 1))) % p
    - Consider x^0 + x^1 + … + x^((y-1)/2) + x^((y-1)/2 + 1) + … + x^y
      = x^0 + x^1 + … + x^((y-1)/2) + x^((y-1)/2 + 1) * (x^0 + x^1 + … + x^((y-1)/2))
      = x^0 + x^1 + … + x ^((y-1)/2) * (1+ x^((y-1)/2 + 1)
- Also x^y can be calculated using recursion.
- So the overall time complexity is O(log K).

Subtask 3

- Subtask 3 (19 points): 1 <= N <= 5000 and 1 <= K <= 5000
- We can use DP to solve this problem.
- Define dp[i] to be the number of ways to put items into the bag with capacity i.
- Then dp[0] + dp[1] + … + dp[K] is the answer we want.
- Base case is dp[0] = 1.
- And the transition formula is dp[i] = $\sum_{j=1}^{N} dp[i - w_j]$ for $i \geq w_j$
- Time complexity: O(NK)
- Remember to take mod at every step.

Subtask 4

- Subtask 4 (13 points): 1 <= K <= 5000
- In addition to subtask 3, although N can be large, but 1 <= w_i <= 100
- So we only need to know the distribution of the weights.
- Let c[i] be the number of items with weight i.
- We can reformulate the formula to

$$dp[i] = \sum_{j=1}^{\max w} c[j] \times dp[i-j] \text{ for } i \geq j$$

- Time complexity = O(K max w_i)

Subtask 5

- Subtask 5 (37 points): No additional constraints
- Reformulate the formula to

$$dp[i] = dp[i-1] + \sum_{j=1}^{100} c[j] \times (dp[i-j] - dp[i-j-1]) \text{ for } i > j$$

$$= dp[i-1] \times (1 + c[1]) + \sum_{j=2}^{100} dp[i-j] \times (c[j] - c[j-1]) - dp[i-101] \times c[100]$$

- Then we can have dp[K] alone as our final answer.
- However, K is too large for us to calculate dp[i] one by one.
- Noted that above is a linear recurrence relationship.
  - We can rewrite it is matrix form.

Subtask 5

- Let v_i be the vector (dp[i], dp[i-1], dp[i-2], … , dp[i-100])
  - set dp[j] = 0 for j < 0
  - v_0 = (1, 0, …, 0)
- Then we have the following: (compare the first row to the formula)

$$v_{i+1} = \begin{pmatrix} c[1]+1 & c[2]-c[1] & \cdots & c[100]-c[99] & -c[100] \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} v_i$$

- By letting A to be the matrix, we have $v_i = A^i v_0$

Subtask 5

- Then we can calculate A^i similar to what we do for number.
  - Please refer to the power point for “Math in OI(II)”
- After all, our answer is just the first entry of v_K
- No. of matrix multiplication: O(log K)
- No. of operation for every matrix multiplication: $O((\max w)^3)$
- Time complexity: $O((\max w)^3 \log K)$

M1742 Number of Spanning Trees

Alex Tung

Statement

- Given a cactus graph, count its number of cycles, modulo $R = 10^9 + 7$

- Simple? ☺

SUBTASKS

For all cases: $1 \leq N \leq 50000$, $N - 1 \leq M \leq 2N - 1$.

	Points	Constraints
1	21	$1 \leq N \leq 10$
2	28	$M = N$ (The graph contains exactly one cycle)
3	16	$1 \leq N \leq 1000$
4	35	No additional constraints

Subtask 1

- Try all $\binom{M}{N-1}$ ways of choosing N – 1 edges
- These edges form a spanning tree iff they connect all N nodes
- Run a dfs (or use other methods) to check whether these edges give a connected graph

Time complexity: $O(2^M * M)$

Subtask 2

- Maintain the degree of the nodes, remove a node X if it becomes a leaf (i.e. when deg[X] = 1)
- The remaining nodes form *the* only cycle of the graph
- The spanning trees of the graph are precisely those formed by removing one edge from the cycle

Time complexity: O(N)

Subtasks 3 - 4

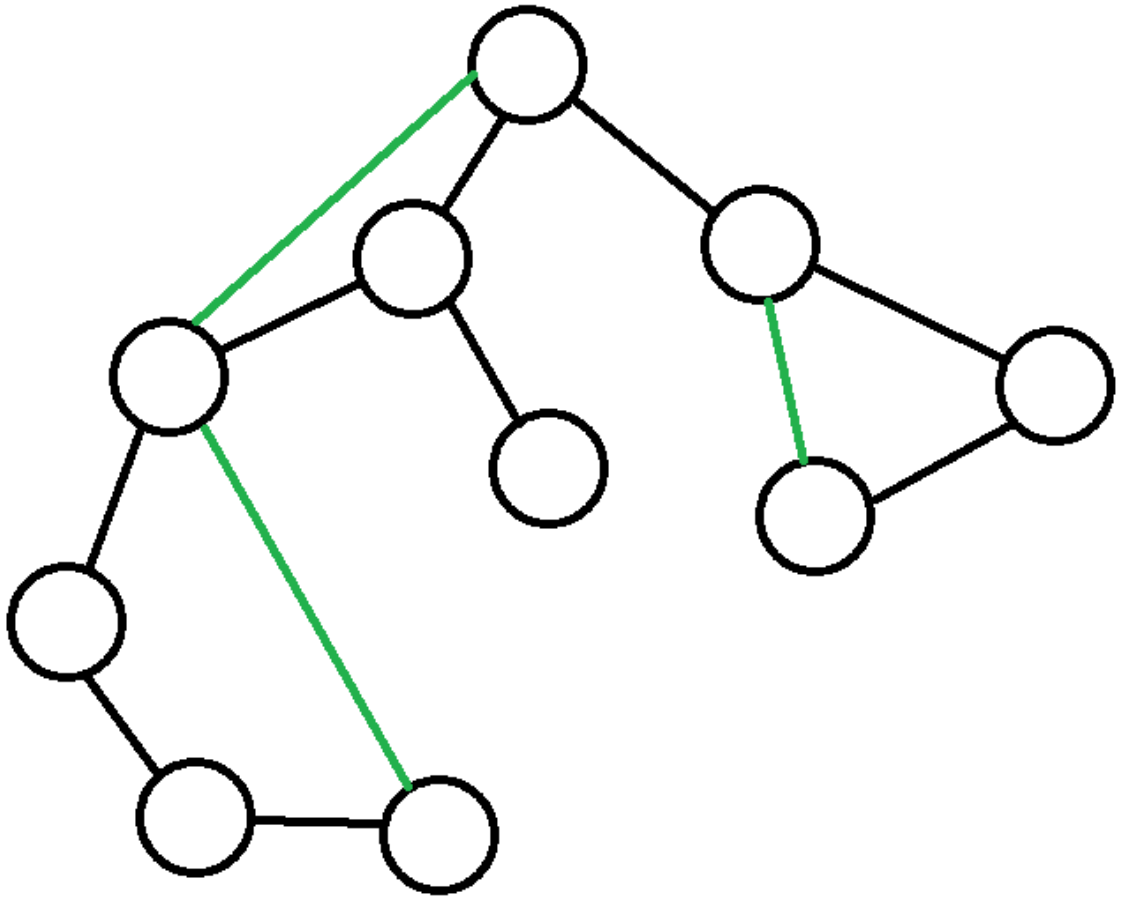
Observation: Let C_1, C_2, ..., C_k be (disjoint) cycles of the graph. Then, answer = $|C_1| * |C_2| * ... * |C_k| \mod R$

Reason: a subgraph of G is a spanning tree if and only if it is formed by removing exactly one edge from each cycle

- Run dfs on the given graph; identify the tree edges and the back edges
- Maintain also the depth of each node

- Note that each cycle corresponds to one back edge

- For each back edge (u_i, v_i), multiply answer by $(dep[u_i] - dep[v_i] + 1)$



“Note that each cycle corresponds to one back edge”

Time complexity:

- $O(N^2)$ for adjacency matrix
- O(N) for adjacency linked list / edge list

M1743 Tree Recovery II

Problem by Charlie Li

Problem statement

- Alice will be given a tree, she need to encode the tree in 01-string.
- Bob will be given the preorder of the tree and the 01-string, he need to decode the 01-string to obtain the postorder of the tree

Scoring

- For all test cases, 1 <= N <= 50000.

- 32 points for length 800000
- 96 points for length 100000
- 97 points for length 99999
- 98 points for length 99998
- 99 points for length 99997
- 100 points for length 99996

Observation

- A preorder and a postorder is enough to uniquely describe a tree (including the order of children)

- So we can only encode the exact tree (including the order of children) or the postorder.

Solution 1

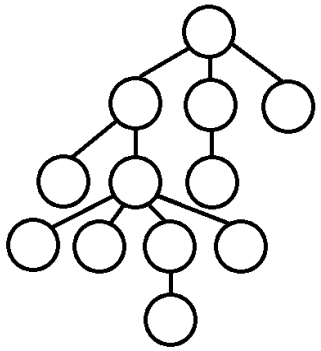
- We can see that the number of nodes are only $50000 < 65536 = 2^{16}$
- So we can encode the postorder of the tree using 16 bits for every entry.

- Eg. Postorder: 1 3 2
  - -> 0000000000000001 0000000000000010 0000000000000011

- Then in worst case, we can use 800000 bits to encode the postorder
  - -> We can get 32 points ☺

Observation 2

- Since the preorder will be given to Bob.
- Perhaps we do not need to encode the whole postorder but only the structure of the tree since the preorder will tell us how to label it.
- Example of “structure”:



- Once we have the structure, we can use the preorder to label it.
- But how can we store the structure?

Observation 2

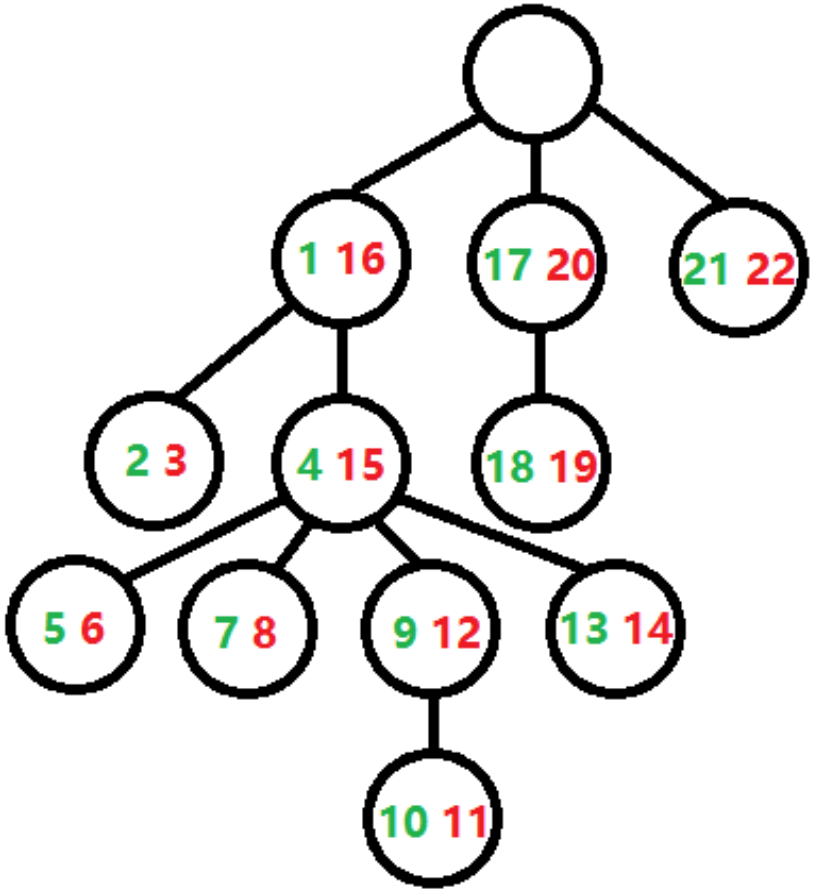
- Actually, there are at least two ways to store the structure
  1. The sequence of going up and down when travelling the graph using dfs (or the sequence of birth and death of the nodes)
  2. The number of children of every node in either preorder or in label order

Solution 2.1

- The picture on the right hand side shows the sequence of going up and down, the green numbers are going down while the red numbers are going up

- Then this graph should give us the following 01-string (0 for down, 1 for up):
  - 00100101001101110011101

- Length: 2(N - 1)



Solution 2.1

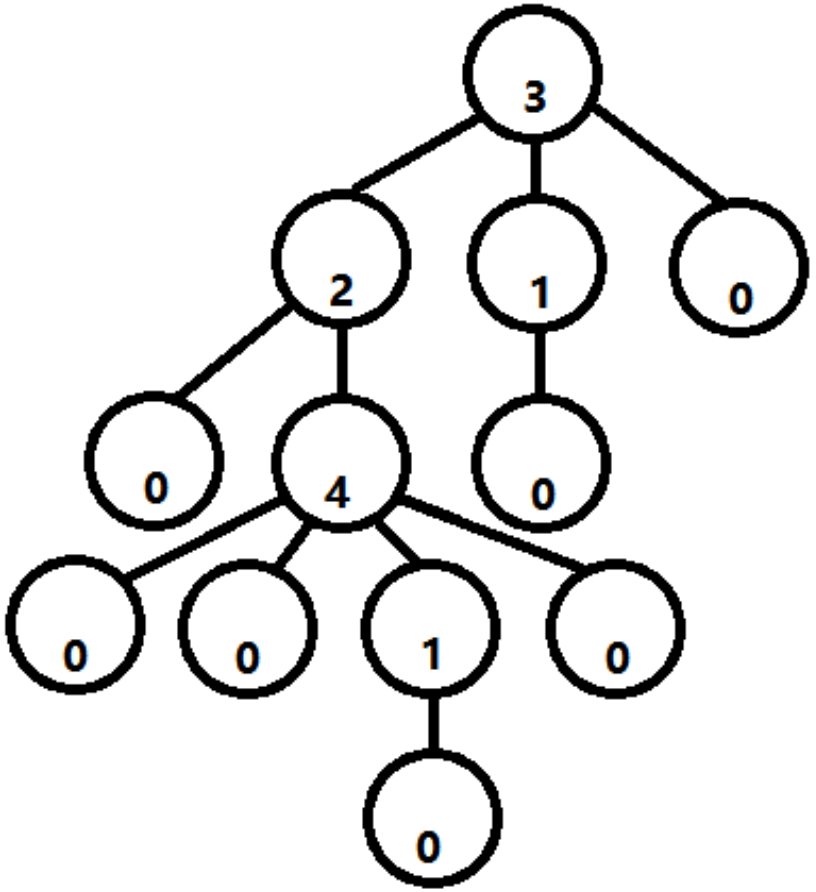
- Decoding (l is a global variable):

```
void dfs() {
    int x;
    scanf("%d", &x);                  // get preorder
    while (s[l++] == '0') dfs();      // travel to children
    printf("%d ", x);                 // output postorder
}
```

- Score: 98 😐

Solution 2.2

- The picture on the right hand side shows the number of children of every node.
- We need to encode it to 01-string, one trivial way is to send 1s followed by a 0.
- Then this graph should give us the following 01-string (in preorder):
  - 1110 110 0 11110 0 0 10 0 0 10 0 0
- Length: 2N – 1, Score: 97 😐



Solution 2.2

- Decoding (l is a global variable):

```
void dfs() {
    int x, y = 0;
    scanf("%d", &x);                  // get preorder
    while (s[l++] == '1') y++;        // count number of children
    for (int i = 0; i < y; i++) dfs(); // travel to the children
    printf("%d ", x);                 // output postorder
}
```

- Score: 97 😐

Observation 3

- If N = 1, we have nothing to do with, just output the preorder as answer. So we can now assume N > 1.
- For solution 2.1, the first character must be 0 and the last one must be 1. So the length can be shortened to 2N – 4. Score: 100 ☺
  - The first character is going down from root
  - The last character is going up back to root

- For solution 2.2, the first character must be 1 and the last two mush be 00. So the length can be shortened to 2N – 4. Score: 100 ☺
  - The first node(root) must have at least one children
  - The last node must be a leaf and have no children

Challenge

- Can you do better then 2N – 4?

- Author’s best: 2N – 5.

M1744 Binary Search Tree

Problem by Charlie Li

Problem statement

- Given a sequence of distinct numbers, construct the binary search tree using the naïve method.

- Naïve method of inserting an element *a*:
  - If current node is empty, insert *a* here.
  - Otherwise if *a* is smaller than the data of current node, go to left subtree.
  - Otherwise, go to right subtree.

Subtasks

- For all cases, $1 <= N <= 500000$, $1 <= a_i <= 10^9$ for $1 <= i <= N$
- Subtask 1 (35 points): $1 <= N <= 1000$
- Subtask 2 (65 points): No additional constraints

Subtask 1

- Subtask 1 (35 points): 1 <= N <= 1000
- For this subtask, what we need to do is just implement the naïve inserting.
- One of the implementation:

```
void insert(int a, node* cur)
    if cur is empty
            cur -> data = a
    else if a < cur -> data
            insert (a, cur -> left)
    else
            insert (a, cur -> right)
```

Subtask 1

- Time complexity: $O(n^2)$

Observation

- To insert an element *a*, it must be inserted as either
  1. the right child of the largest number smaller than *a.*
  2. the left child of the smallest number larger than *a*.
- Also, there will be exactly one of them is empty.

- Why? (Try to work it out yourself)

Subtask 2

- Subtask 2 (65 points): No additional constraints
- For every new element, we only need to find the two closest elements.

- For c++ users, std::set<int> and lower_bound() for set is extremely useful.
- If you are not using c++, you may need to write a segment tree to do so.

Subtask 2

- How to use segment tree to find largest number smaller than *a*?
- First, let b[i] be the position of i in a[].
- We first build a segment tree from 1 to n where every node stores the maximum value in the range. Initially, all are zero.
- Then starting from i, up to n
  - Query the maximum in the range 1 to b[i]
    - If this is zero, than there are no number smaller than i.
    - Otherwise, this is the largest number smaller than i.

- To find the smallest number larger than *a*, we can use the above strategy to build another segment tree which stores minimum value and reverse the order of query.