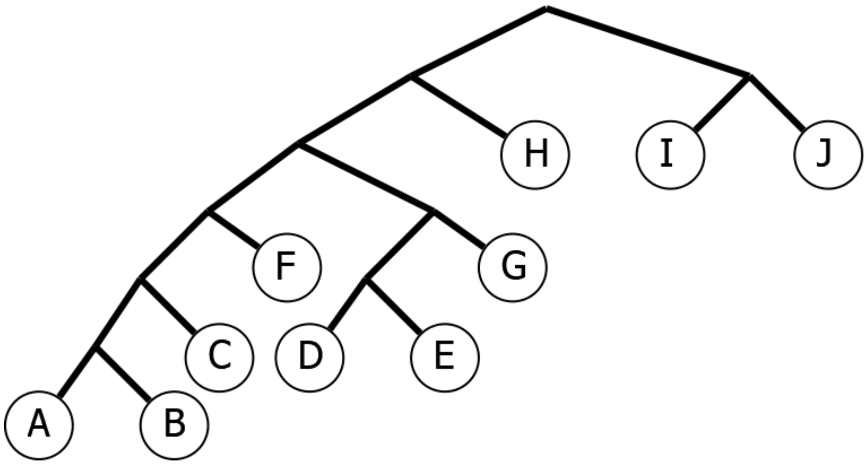
Question 1 (Huffman Tree, 30 points). *Give a tree for the optimal Huffman code for the following letters and frequencies:*
$A \times 1, B \times 5, C \times 13, D \times 15, E \times 17, F \times 25, G \times 36, H \times 100, I \times 130, J \times 170.$

The solution is given below:



This can be obtained using the standard Huffman code algorithm. A and B are combined to create a new node AB with weight 6. This is then combined with C to create a node of weight 19. D and E are combined to create a node of weight 32. Then ABC is combined with F to create a node of weight 44. DE is combined with G to create a node of weight 68. DEG is combined with $ABCF$ to create a node of weight 112, which is then combined with H to create a node of weight 212. Next, I and J are combined to create a node of weight 300. Finally, IJ is combined with $ABCDEFGH$ to create a node of weight 512.

Question 2 (No Spaces, 35 points)**.** *If sentences are written without spaces, there can be ambiguity about how to interpret them. For example, "`HEADDRESSES`" could be either "`HEAD DRESSES`" or "`HE ADDRESSES`". Give an algorithm that given a vocabulary list of k words (given as strings of characters) and a text string of n characters determines the number of ways to break the text up into some number of words from the vocabulary. For full credit, your algorithm should run in time polynomial in n and k.*

The algorithm is as follows:

```
Interpretations(Text T)
  Create array I[0..n]
    \\ I[m] will store the number of ways to break up the first m letters of the text
  I[0] = 1
  For m = 1 to n
    I[m] = 0
    For s = 1 to m
      If T[s],T[s+1],...,T[m] is in the dictionary
        I[m] = I[m] + I[s-1]
  Return I[n]
```

The runtime here is clearly polynomial. The loop over m has $O(n)$ iterations, as does the loop over s. The check to see whether $T[s]\ldots T[m]$ is in the dictionary takes at most $O(kn)$ time, so the final runtime is $O(kn^3)$.

To show correctness, we claim that at the end of the $m-th$ iteration of the outer loop, $I[m]$ stores the number of interpretations of the first m characters of the text. Firstly, $I[0]$ is correctly assigned to 1 (as the empty string can be obtained as a list of no words). The rest we prove by strong induction on m. Assuming that all previous $I[s-1]$ have been correctly computed, $I[m]$ ends up as the sum over all s so that $I[s]\ldots I[m]$ is a word of $I[s-1]$. This counts the number of interpretations of the first m characters where the last word in the string is $I[s]\ldots I[m]$. Summing over all s gives the correct result.

Note: By converting our dictionary into a trie, we can do the dictionary lookups much faster, and an optimized version of this algorithm will have runtime $O(kn+n^2)$.

Question 3 (Business Plan, 35 points)**.** *Karkat owns a factory that makes* \$1000*/year. However, he has plans to expand it. He has k possible expansion options. Each has a cost* C_i *and would multiply his factory's yearly profit by a multiple* M_i*. He can build at most one new expansion per year, and only if he has the money available (he starts with* \$0*). However, the improvements from multiple expansions will stack with each other (though each improvement can only be built once). So, for example, if he has one improvement that multiplies his profit by a factor of 2 and another that multiplies it by a factor of 3, his profit for that year will be* \$6000*.*

Karkat wants to make a plan for this factory over the course of the next n years. Give an algorithm to compute the greatest amount of money he could end up with by the end of this period. For full credit, your algorithm should run in time $O(nk2^k)$ *or better.*

Note: you only need to compute the amount of money, not the strategy that Karkat uses to obtain it.

We create a dynamic program where our subproblem asks for the most amount of money that Karkat could have at the end of year r while having exactly a particular set S of expansions.

The algorithm is as follows:

```
BestStrategy(n,Options)
  Create a table T indexed by an integer 0 <= r <= n
    and a subset S of the options
  Let T[0,emptyset] = 1
  Let T[0,S] = -Infinity for each nonempty S
  For r = 1 to n
    For S subset of options
      T[r,S] = T[r-1,S] + $1000*(product of multipliers in S)
      For options O in S
        Val = T[r-1,S-O]+$1000*(products of mult. in S-O)-cost(O)
        If Val > 0
          T[r,S] = max(T[r,S],Val)
  Return maximum over S of T[n,S]
```

To show correctness we claim that each $T(r, S)$ is assigned the correct answer to its corresponding subproblem (or $-\infty$ if Karkat cannot have that set of improvements by that year). For $r = 0$, this is clear as he must start year 0 with \$0 and no expansions. For larger r, we proceed by induction on r.

In order to have exactly the expansions in S by the end of year r, Karkat must either have had exactly S at the end of year $r - 1$ or had $S - O$ for some option $O \in S$. In the former case, Karkat would have the amount of money he had at the end of year $r - 1$ (at most $T[r - 1, S]$), plus his earnings of \$1000 times the product of the multipliers in S. In the latter case, Karkat would have had at most $T[r - 1, S - O]$ money at the end of year $r - 1$. He would have earned \$1000 times the product of the multipliers in $S - O$ in year r, and if this is more than the cost of O would have ended up with the difference at the end of year r. The algorithm correctly computes the largest amount of money Karkat could have had from any of these options and sets it to $T[r, S]$.

As for the runtime, we have $O(2^k n)$ subproblems. To fill out each entry of the table, we need to check $O(k)$ different values of O. If we have precomputed the products of the multipliers for each set S (which can be done in $O(k2^k)$ time), each check takes $O(1)$ time, for final runtime of $O(nk2^k)$.