```
void knapsack (index i,
               int profit, int weight)
{ counter++
  if (weight <= W && profit > maxprofit){          // This set is b
     maxprofit = profit;                           // so far.
     numbest = i;                                  // Set numbest
     bestset = include;                            // number of ite
  }                                                // considered. S
   (i, #), profit, weight, maxprof,                // bestset to t
                                                   // solution.
  if (promising(i)){
     include[i + 1] = "yes";                       // Include w[i+
     knapsack(i + 1, profit + p[i + 1], weight + w[i + 1]);
     include[i + 1] = "no";                        // Do not inclu
     knapsack(i + 1, profit, weight);              // w[i + 1].
  }
} else backtrack

bool promising (index i)  & int profit, int weight
{
  index j, k;
  int totweight;
  float bound;

  if (weight >= W)                                 // Node is promising only
     return false;                                 // if we should expand t
  else{                                            // its children. There mus
     j = i + 1;                                    // be some capacity left fo
     bound = profit;                               // the children.
     totweight = weight;
     while (j <= n && totweight + w[j] <= W){
                                                   // Grab as many items as
       totweight = totweight + w[j];               // possible.
       bound = bound + p[j];
       j++;
     }
     k = j;                                        // Use k for consistency
     if (k <=n)                                    // with formula in text.
       bound = bound + (W - totweight) * p[k]/w[k];
     bound                                         // Grab fraction of kth
     return bound > maxprofit;                     // item.
  }
}
```