Problem Solving & Software Development

Lecture 4. Recursion and Backtracking

Subset Sum Problem

Given an array of numbers and a target value, find whether there exists a subset of those numbers that sum up to the target value.

```
boolean subsetSum (int[] a, int target)
```

Subset Sum Problem

Given an array of numbers and a target value, find whether there exists a subset of those numbers that sum up to the target value.

```
boolean subsetSum (int[] a, int target)
```

Recursive structure:

- consider the next element in the array
- try making a sum WITH this element
- try making a sum WITHOUT this element
- if neither is possible, return false

Subset Sum Problem

- Consider the next element, it is either in the solution, or not.
- Try both ways. If both fail, return false.

- Need to keep track of the partial sum so far.
- When starting a recursive call, need the sum of the current subset.
- Also need to know the index of the next element to consider.

```
void recSubset(int[] a, int target, int i, int sumSoFar)
```

Need a wrapper:

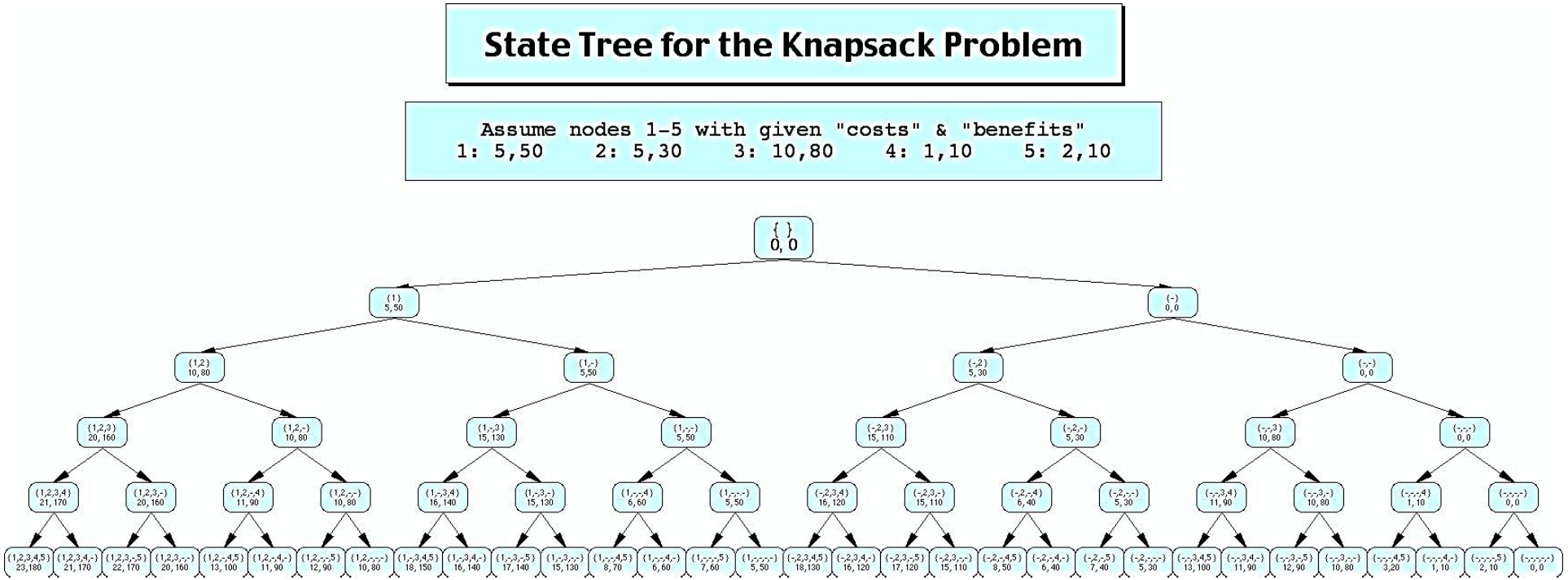
```
boolean subsetSum (int[] a, int target) {
   return recSubset(a, target, 0, 0);
}
```

Subset Sum Problem

```
// i is the index of the next element to consider
// sumSoFar is the sum of elements included in the solution so far

boolean recSubset(int[] a, int target, int i, int sumSoFar) {

   // base cases
   if (sumSoFar == target) return true;

   // we reached the end and sum is not equal to target
   if (i == a.length) return false;

   //recursive case: try next element both in and out of the sum
   boolean with = recSubset(a, target, i+1, sumSoFar + a[i]);
   boolean without = recSubset(a, target, i+1, sumSoFar);
   return (with || without);

}
```

Knapsack Problem

The goal of the knapsack problem is to choose a subset of given items that maximizes total **benefit** while staying within the limit of a prescribed total **cost**.



Permutations

Write a function to print all permutations of a given string.

Example: permute “abc” should print: abc, acb, bca, bac, cab, cba.

```
void printPerm(String s)
```

Recursive structure:

- Chose a letter from the input, and make this the first letter of the output
- Recursively permute remaining input

- What is the base case?
- Can you make sure that each permutation is generated precisely once?

Permutations

Idea: pick a letter, add it to the solution, recourse on remaining.

```
// print soFar + all permutations of remaining
void recPermute(String soFar, String remaining)
```

Need a wrapper:

```
// print all permutations of s
void printPerm (String s) {
    recPermute("", s);
}
```

Why use wrappers?

– the user does not need to know the internals of the implementation. In this case, that it is recursive.

Permutations

```
void recPermute(String soFar, String remaining) {

   // base case
   if (remaining.length() == 0)
   System.out.println(soFar);

   // general case
   else {
      for (int i=0; i< remaining.length(); i++) {
      String nextSoFar = soFar + remaining[i];
      String nextRemaining =  remaining.substring(0,i) +
                              remaining.substring(i+1);
      recPermute(nextSoFar, nextRemaining);
      }
   }
}
```

Assignment Problem

Assume that we have **N workers** and **N jobs** that should be done.

For each pair (worker, job) we know salary that should be paid to worker for him to perform the job.

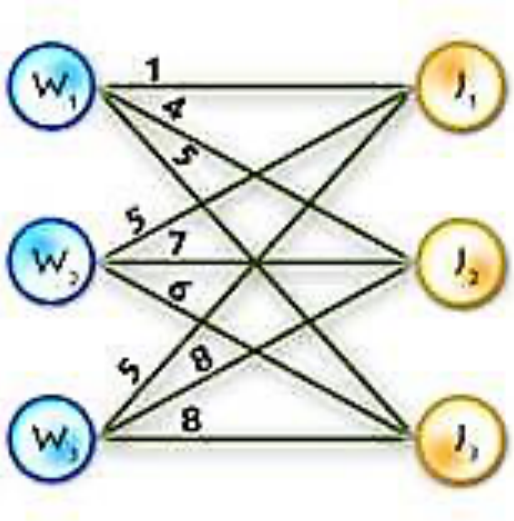
Our goal is to complete all jobs minimizing total costs,
while assigning each worker to exactly one job and vice versa.

jobs

workers

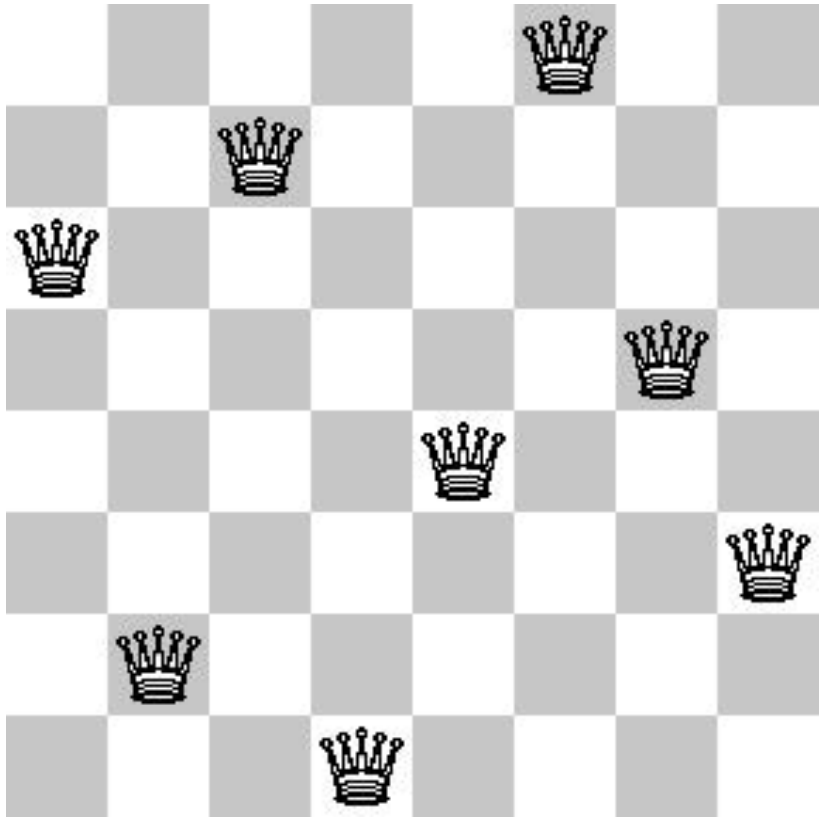
$$\begin{pmatrix} 1 & 4 & 5 \\ 5 & 7 & 6 \\ 5 & 8 & 8 \end{pmatrix}$$

$\Leftrightarrow$



N Queens Problem

How can N queens be placed on an NxN chessboard so that no two of them attack each other?



N-Queen using backtracking

- In backtracking you perform recursive search until you find a solution (it doesn't have to be the best solution).

- If you change global variables as you search then you need undo these changes on return.
  - In the N-queen problem the queens' positions are the lgobal variable

8-Queens Code Sketch

```
Global variable: positions = [-1 -1 -1 -1 -1 -1 -1 -1 ]  // row of each queen
bool eightqueens(int col):
      if col=8:   // we have done all cols.
            return true;

            // for each potential row
      for row in 0..7:
         // try placing the queen
         positions[col]=row;
         if not incheck() and eightqueens(col+1)
            return true;

       // solution not found
       positions[col] = -1;  // reset column (backtrackiing)
       return false;

// call
if eightqueens(0):
   print positions;
```

N Queens Problem

How can N queens be placed on an NxN chessboard so that no two of them attack each other?