Introduction to Algorithms

Department of Computer Science and Engineering

East China University of Science and Technology

Back-Tracking Algorithms

Design and Analysis of Algorithms

Back-Tracking Algorithms

Topics:
- •General Method
- •N-Queen problem

Back-Tracking Paradigm

- A design technique, like divide-and-conquer.
- Useful for optimization problems and finding feasible solutions.
  - Solution to a problem is defined as an n-tuple:
    $(x_1, x_2, \ldots, x_n)$, where x_i is taken from a finite set S_i .

  Generally, we need to:
    - finding a vector which maximize (or minimize) a specific objective function $P(x_1, x_2, \ldots, x_n)$.
    - finding a (or all) vector(s) that satisfy a specific criterion function $P(x_1, x_2, \ldots, x_n)$.

Back-Tracking——General Method

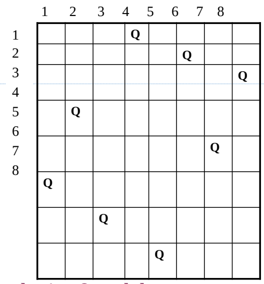
- Constraints
  - Explicit Constraints
    - constrain values of each component x_i;
    - All tuples that satisfy explicit constraints make up a possible solution space.
  - Implicit Constraints:
    - inter-components constraints;
    - Implicit constraints identify those satisfying the criterion function in the solution space.

8-Queen problem

- Place 8 queens on a 8×8 chessboard, yielding that any two of them do not conflict, i.e. any two of them do not reside in the same row, column or diagonal.

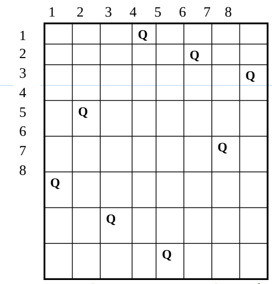
	1	2	3	4	5	6	7	8
1				Q				
2						Q		
3								Q
4		Q						
5							Q	
6	Q							
7			Q					
8					Q			

8-Queen problem



- Label the rows and columns by 1 to 8, and the queens too.
- Assume that queen *i* resides in row *i*.
- Solutions can be defined as a 8-tuple (x_1, x_2, …, x_8), where x_i is the column number of queen i.
- The solution above is (4，6，8，2，7，1，3，5).

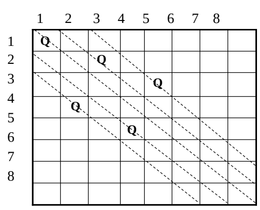
8-Queen problem



- **Explicit Constraints: $x_i \in S_i$， S_i={1,2,3,4,5,6,7,8}, 1⩽ i⩽ 8. The solution space consists of 8^8 8-tuples.**
- **Implicit Constraints: any two of them do not reside in the same row, column(any two of x_i differ) or diagonal.**

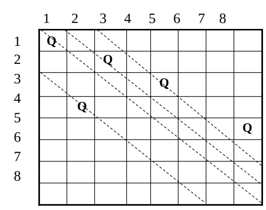
8

8-Queen problem



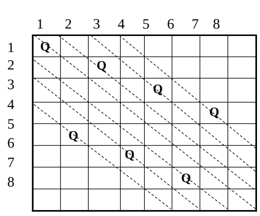
9

8-Queen problem



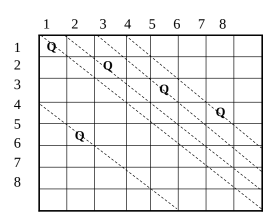
10

8-Queen problem



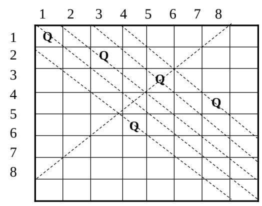
11

8-Queen problem

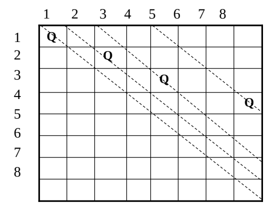


12

8-Queen problem



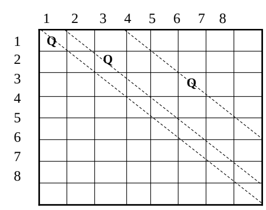
8-Queen problem



Simple Review

- All-pairs shortest paths
  - Floy-Warshall algorithm
    - $c_{ij}^{(k)}=\min_k\{c_{ij}^{(k-1)}, c_{ik}^{(k-1)}+c_{kj}^{(k-1)}\}$
  - Johnson's algorithm
    - Graph reweighting
      - $h: V \rightarrow R$, reweight $(u,v) \in E$ by $w_h(u,v)=w(u,v)+h(u)-h(v)$.
    - Algorithm:
      - Find a function $h: V \rightarrow R$, such that $w_h(u,v) \geq 0$ for all $(u,v) \in E$ by using Bellman-Ford
      - using w_h from Run Dijkstra's algorithms each vertex $u \in V$
      - For each $(u,v) \in V*V$, compute$\delta(u,v)=\delta_h(u,v)-h(u)+h(v)$
- Back-Tracking Paradigm

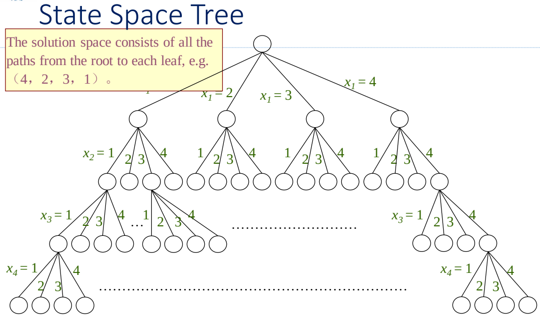
8-Queen problem



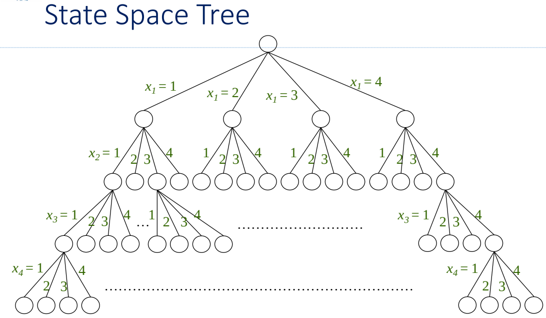
Back-Tracking Paradigm finds answers by systematic searching the solution space of a given problem.

State Space Tree

The solution space consists of all the paths from the root to each leaf, e.g.（4，2，3，1）。

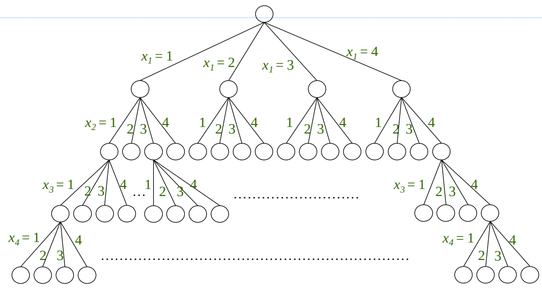


State Space Tree



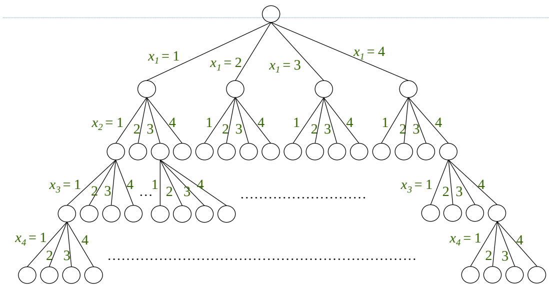
- Problem State
  - Each node in the tree identifies a problem state when solving the problem.

State Space Tree



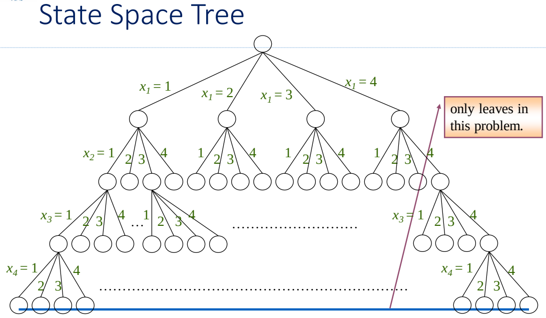
- State Space
  - All the paths from the root to each node

State Space Tree



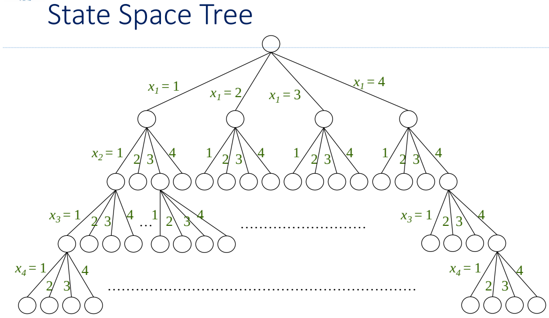
- State Space Tree
  - The above tree of the solution space

State Space Tree



- Solution States
  - Those problem states to which the path from the root identifies a vector in the solution space (Satisfying explicit constraints).

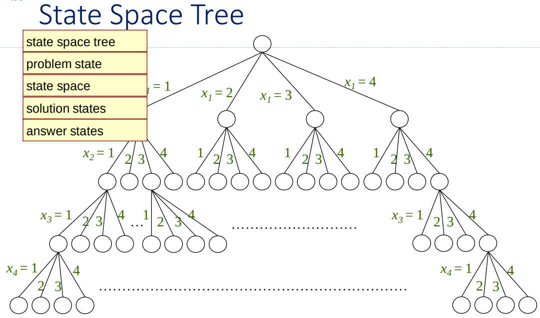
State Space Tree



- Answer States
  - Those solution states to which the path from the root identifies an answer (Satisfying implicit constraints).

State Space Tree

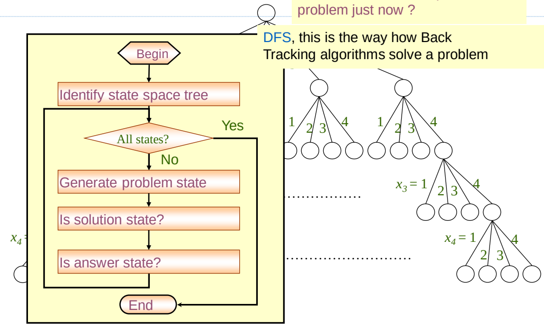
- state space tree
- problem state
- state space
- solution states
- answer states



State Space Tree

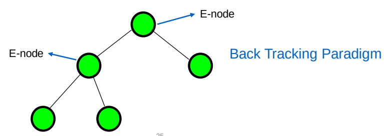
how did we solve the queen problem just now ?

DFS, this is the way how Back Tracking algorithms solve a problem



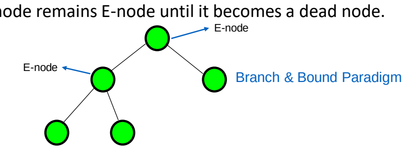
Searching in State Space Tree

- Basic Concepts
  - Alive node: a generated node whose sons have not all been generated.
  - E-node: the node whose son is generating currently.
  - Dead node: a node that all his sons have been generated or there is no need to generate his sons.
- Generating problem states in DFS way
  - Once a son C of the E-node R is generated, the generated son C become the new E-node, node R will become E-node again after all the tree rooted by node C is checked.

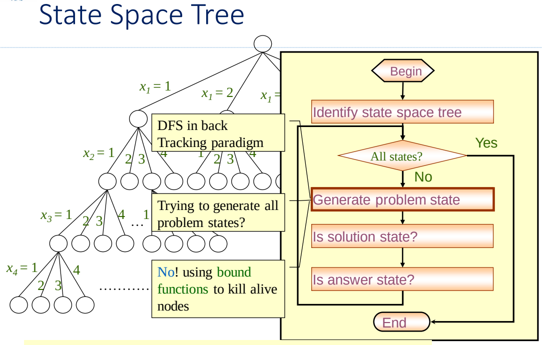


Searching in State Space Tree

- Basic Concepts
  - Alive node: a generated node whose sons have not all been generated.
  - E-node: the node whose son is generating currently.
  - Dead node: a node that all his sons have been generated or there is no need to generate his sons.
- Generating problem states in BFS way
  - A E-node remains E-node until it becomes a dead node.

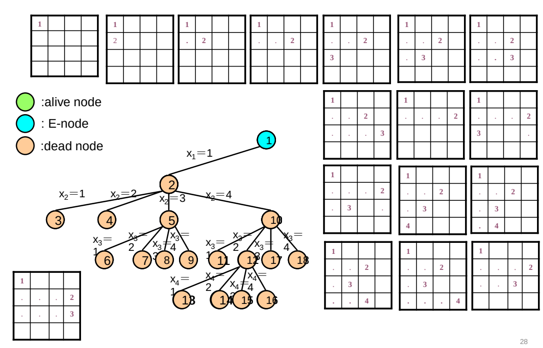


State Space Tree

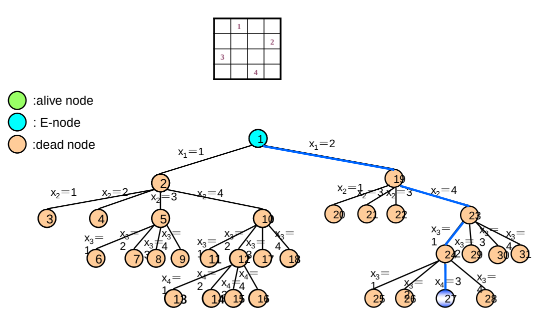


This is how Back-Tracking varies from Brute-Force

4-Queen problem



4-Queen problem



General Method of Back Tracking

BACKTRACKING

```
BACKTRACKING(n)
 k←1;
 while k>0 do
   if there are unchecked X(k), X(k)∈T(X(1),…,X(k-1)) and B(X(1),…,X(k))=true
      then   if (X(1),…,X(k) is an answer）
                then  print (X(1),…,X(k))
             if (k < n)
                k←k+1
      else
             k←k-1
```

- Construct solution in X(1:n), an answer is output immediately after it is determined.
- T(X(1),...,X(k-1)) : returns all possible X(k) ,given X(1),...,X(k-1).
- B(X(1),...,X(k)): returns whether X(1),...,X(k) satisfies the implicit constraints.

General Method of Back Tracking

```
BACKTRACKING
BACKTRACKING(n)
 k←1;
 while k>0 do
  if there are unchecked X(k), X(k)∈T(X(1),…,X(k-1)) and B(X(1),…,X(k))=true
     then  if (X(1),…,X(k) is an answer)
              then  print (X(1),…,X(k))
           if (k < n)
              k←k+1
     else
          k←k-1
```

Whether there is any son that has not been generated for the current node. Bound function is used here.

- Construct solution in X(1:n), an answer is output immediately after it is determined.
- T(X(1),...,X(k-1)) : returns all possible X(k) ,given X(1),...,X(k-1).
- B(X(1),...,X(k)): returns whether X(1),...,X(k) satisfies the implicit constraints.

General Method of Back Tracking

```
BACKTRACKING
BACKTRACKING(n)
 k←1;
 while k>0 do
  if there are unchecked X(k), X(k)∈T(X(1),…,X(k-1)) and B(X(1),…,X(k))=true
     then  if (X(1),…,X(k) is an answer)
              then  print (X(1),…,X(k))
           if (k < n)
              k←k+1
     else
          k←k-1
```

If the new generated node is an answer state, print the answer.

- Construct solution in X(1:n), an answer is output immediately after it is determined.
- T(X(1),...,X(k-1)) : returns all possible X(k) ,given X(1),...,X(k-1).
- B(X(1),...,X(k)): returns whether X(1),...,X(k) satisfies the implicit constraints.

General Method of Back Tracking

```
BACKTRACKING
BACKTRACKING(n)
 k←1;
 while k>0 do
  if there are unchecked X(k), X(k)∈T(X(1),…,X(k-1)) and B(X(1),…,X(k))=true
     then  if (X(1),…,X(k) is an answer)
              then  print (X(1),…,X(k))
           if (k < n)
              k←k+1
     else
          k←k-1
```

DFS: the new generated node becomes the E-node.

- Construct solution in X(1:n), an answer is output immediately after it is determined.
- T(X(1),...,X(k-1)) : returns all possible X(k) ,given X(1),...,X(k-1).
- B(X(1),...,X(k)): returns whether X(1),...,X(k) satisfies the implicit constraints.

General Method of Back Tracking

```
BACKTRACKING
BACKTRACKING(n)
 k←1;
 while k>0 do
  if there are unchecked X(k), X(k)∈T(X(1),…,X(k-1)) and B(X(1),…,X(k))=true
     then  if (X(1),…,X(k) is an answer)
              then  print (X(1),…,X(k))
           if (k < n)
              k←k+1
     else
          k←k-1
```

All sons have been checked, Tracking back.

- Construct solution in X(1:n), an answer is output immediately after it is determined.
- T(X(1),...,X(k-1)) : returns all possible X(k) ,given X(1),...,X(k-1).
- B(X(1),...,X(k)): returns whether X(1),...,X(k) satisfies the implicit constraints.

Back-Tracking Algorithm for n-Queen problem

```
NQUEENS
NQUEENS(n)
   X(1)←0;k←1
   while k>0 do
       X(k)←X(k)+1
       while X(k) ≤ n and not PLACE(k) do
          X(k)←X(k)+1
       if X(k) ≤ n
          then   if k=n
                    then print (X)
                    else k←k+1;X(k)←0
          else k←k-1;
```

k : current row;
X(k) : column NO of queen k

Finding next valid column NO using bound function PLACE.

Found or not?

Found! Is it an answer? Print the answer, or shift to the next row and find a place for next queen.

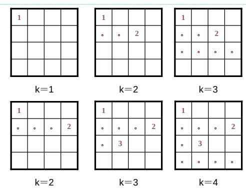
No! Tracking back.

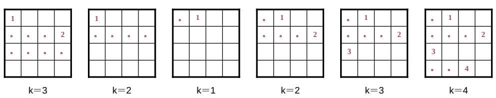
Bound function -- PLACE

```
NQUEENS
PLACE(k)
  i←1
  while i<k do
      if X(i)=X(k) or |X(i)-X(k)| = |i-k|
          then return false
      i←i+1
  return true
```

4-Queen Problem

```
NQUEENS(n)
   X(1)←0;k←1
   while k>0 do
        X(k)←X(k)+1
        while X(k) ≤ n and not PLACE(k) do
            X(k)←X(k)+1
        if X(k) ≤ n
            then   if k=n
                      then print (X);
                      else k←k+1;X(k)←0
            else k←k-1;
```





Further Reading

- Further reading:
  - O(1) time PLACE.
  - Generate answers for n-queen problem w/o searching.

Design and Analysis of Algorithms

Back-Tracking Algorithms

Topics:
- **Subset-sum problem**

General Method of Back Tracking

BACKTRACKING

```
BACKTRACKING(n)
 k←1;
 while k>0 do
   if there are unchecked X(k), X(k)∈T(X(1),…,X(k-1)) and B(X(1),…,X(k))=true
      then   if (X(1),…,X(k) is an answer)
                  then   print (X(1),…,X(k))
              if (k < n)
                  k←k+1
      else
              k←k-1
```

- Construct solution in X(1:n), an answer is output immediately after it is determined.
- T(X(1),…,X(k-1)) : returns all possible X(k) ,given X(1),…,X(k-1).
- B(X(1),…,X(k)): returns whether X(1),…,X(k) satisfies the implicit constraints.

Recursive Method of Back Tracking

RECUSIVE-BACKTRACKING

```
RECURSIVE-BACKTRACKING( k )
   for each X(k), X(k)∈T(X(1),…,X(k-1)) and B(X(1),…,X(k))=true do
        if (X(1),…,X(k) is an answer)
               then  print (X(1),…,X(k))
        if (k<n)
               then call RECURSIVE-BACKTRACKING( k+1 )
```

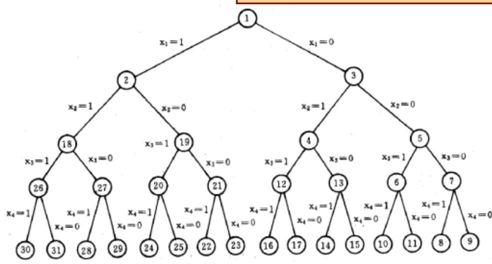
- Construct solution in X(1:n), an answer is output immediately after it is determined.
- T(X(1),…,X(k-1)) : returns all possible X(k) ,given X(1),…,X(k-1).
- B(X(1),…,X(k)): returns whether X(1),…,X(k) satisfies the implicit constraints.

Subset-sum problem

- Given *n+1* positive integers: w_i, 1≤i≤n, and M , Find all the subsets of $W=\{w_i\}$, of which the summary equals to M.
  - E.g. n＝4, $(w_1, w_2, w_3, w_4)=(11, 13, 24, 7)$, $M=31$
    - The expected subsets are (11，13，7) and (24，7).
- The form of solution
  - A solution of subset-sum problem is defined as an n-tuple $(x_1, x_2, \ldots, x_n)$, where $x_i \in \{0, 1\}$, 1≤i≤n. If w_i is included in the subset, then $x_i=1$, otherwise $x_i=0$.
    - The above answers can be defined as （1，1，0，1）and（0，0，1，1）。

State space tree for Subset-sum problem

It is easy to determine X(i)/generate problem state.
The key is the bound function.

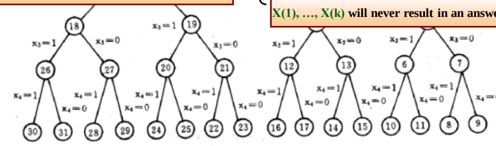


Bound function

Sort W(i) in increasing order.
Bound function is: B(X(1),…, X(k))=true
iff $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \ge M$ and
$\sum_{i=1}^{k} W(i)X(i) + W(k+1) \le M$

A simple choice is : B(X(1),…, X(k))=true
iff $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \ge M$

if W(i) are sorted in increasing order, and if
$\sum_{i=1}^{k} W(i)X(i) + W(k+1) > M$
X(1), …, X(k) will never result in an answer.



Recursive Method of Bac…

Sort W(i) in increasing order.
Bound function is: B(X(1),…, X(k))=true
iff $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \ge M$ and
$\sum_{i=1}^{k} W(i)X(i) + W(k+1) \le M$

RECUSIVE-BA…

{0 … }

RECURSIVE-BACKTRACKING(*k* **)**
  for each X(k), X(k)∈T(X(1),…,X(k-1)) and B(X(1),…,X(k))=true do
    if (X(1),…,X(k) is an answer）
      then print (X(1),…,X(k))
    if (k<n)
      then call RECURSIVE-BACKTRACKING(*k+1*)

- Construct solution in X(1:n), an answer is output immediately after it is determined.
- T(X(1),…,X(k-1)) : returns all possible X(k) ,given X(1),…,X(k-1).
- B(X(1),…,X(k)): returns whether X(1),…,X(k) satisfies the implicit constraints.

Back Tracking Algorithm for S…

Original Recursive Algorithm:

Sort W(i) in increasing order.
Bound function is: B(X(1),…, X(k))=true
iff $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \ge M$ and
$\sum_{i=1}^{k} W(i)X(i) + W(k+1) \le M$

ORIGINAL-…

ORIGINAL-SUMOFSUB(*k*)
  X(k)←1
  if $\sum_{i=1}^{k} W(i)X(i) = M$
    then print(X(j),j←1 to k)
    else if $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \ge M$ and $\sum_{i=1}^{k} W(i)X(i) + W(k+1) \le M$
      then call ORIGINAL-SUMOFSUB(k+1)
  X(k)←0
  if $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \ge M$ and $\sum_{i=1}^{k} W(i)X(i) + W(k+1) \le M$
    then call ORIGINAL-SUMOFSUB(k+1)

Back Tracking Algorithm for S…

Sort W(i) in increasing order.
Bound function is: B(X(1),…, X(k))=true
iff $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \ge M$ and
$\sum_{i=1}^{k} W(i)X(i) + W(k+1) \le M$

SUMO…

SUMOFSUB(*s,k,r*)
  X(k)←1
  if s+W(k)=M
    then print(X(j),j←1 to k)
    else if s+W(k)+W(k+1) ≤ M
      then call SUMOFSUB(s+W(k),k+1,r-W(k))
  if s+r-W(k) ≥ M and s+W(k+1) ≤ M
    then X(k)←0
      call SUMOFSUB(s,k+1,r-W(k))

Back Tracking Algorithm for S…

Sort W(i) in increasing order.
Bound function is: B(X(1),…, X(k))=true
iff $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \ge M$ and
$\sum_{i=1}^{k} W(i)X(i) + W(k+1) \le M$

SUMO…

SUMOFSUB(*s,k,r*)
  X(k)←1
  if s+W(k)=M
    then print(X(j),j←1 to k)
    else if s+W(k)+W(k+1) ≤ M
      then call SUMOFSUB(s+W(k),k+1,…
  if s+r-W(k) ≥ M and s+W(k+1) ≤ M
    then X(k)←0
      call SUMOFSUB(s,k+1,r-W(k))

When called, X(1),X(2),…X(k-1) have been determined.
$s = \sum_{j=1}^{k-1} W(j)X(j)$ and $r = \sum_{j=k}^{n} W(j)$
$W(j)$ are sorted in increasing order.
$W(1) \le M, \sum_{i=1}^{n} W(i) \ge M$

Back Tracking Algorithm for

Sort W(i) in increasing order.

Bound function is: B(X(1),…, X(k))=true

iff $\sum_{i=1}^{k} W(i)X(i)+\sum_{i=k+1}^{n} W(i)\ge M$ and

$\sum_{i=1}^{k} W(i)X(i)+W(k+1)\le M$

SUMO

SUMOFSUB(*s,k,r*)
X(k)←1
if s+W(k)=M
then print(X(j),j←1 to k)
else if s+W(k)+W(k+1) ≤ M
then call SUMOFSUB(s+W(k),k+1,r-W(k))
if s+r-W(k) ≥ M and s+W(k+1) ≤ M
then X(k)←0
call SUMOFSUB(s,k+1,r-W(k))

Initial call:

$$SUMOFSUB(\,0,1,\sum_{i=1}^{n} W(i))$$

Back Tracking Algorithm for

Sort W(i) in increasing order.

Bound function is: B(X(1),…, X(k))=true

iff $\sum_{i=1}^{k} W(i)X(i)+\sum_{i=k+1}^{n} W(i)\ge M$ and

$\sum_{i=1}^{k} W(i)X(i)+W(k+1)\le M$

SUMO

SUMOFSUB(*s,k,r*)
X(k)←1
if s+W(k)=M
then print(X(j),j←1 to k)
else if s+W(k)+W(k+1) ≤ M
then call SUMOFSUB(s+W(k),k+1,r-W(k))
if s+r-W(k) ≥ M and s+W(k+1) ≤ M
then X(k)←0
call SUMOFSUB(s,k+1,r-W(k))

Check left son(X(k)=1).

Back Tracking Algorithm for

Sort W(i) in increasing order.

Bound function is: B(X(1),…, X(k))=true

iff $\sum_{i=1}^{k} W(i)X(i)+\sum_{i=k+1}^{n} W(i)\ge M$ and

$\sum_{i=1}^{k} W(i)X(i)+W(k+1)\le M$

SUMO

SUMOFSUB(*s,k,r*)
X(k)←1
if s+W(k)=M
then print(X(j),j←1 to k)
else if s+W(k)+W(k+1) ≤ M
then call SUMOFSUB(s+W(k),k+1,r-W(k))
if s+r-W(k) ≥ M and s+W(k+1) ≤ M
then X(k)←0
call SUMOFSUB(s,k+1,r-W(k))

Print the answer once found. No recursive call after finding an answer.

Back Tracking Algorithm for

?do not check $\sum_{i=1}^{k} W(i)X(i)+\sum_{i=k+1}^{n} W(i)\ge M$

Initail: $s+r\ge M\,(s=0, r=\sum_{i=1}^{n} W(i))$

Sort W(i) in increasing order.

Bound function is: B(X(1),…, X(k))=true

iff $\sum_{i=1}^{k} W(i)X(i)+\sum_{i=k+1}^{n} W(i)\ge M$ and

$\sum_{i=1}^{k} W(i)X(i)+W(k+1)\le M$

SUMO

SUMOFSUB(*s,k,r*)
X(k)←1
if s+W(k)=M
then print(X(j),j←1 to k)
else if s+W(k)+W(k+1) ≤ M
then call SUMOFSUB(s+W(k),k+1,r-W(k))
if s+r-W(k) ≥ M and s+W(k+1) ≤ M
then X(k)←0
call SUMOFSUB(s,k+1,r-W(k))

If the bound function is satisfied, do recursive call.

"s+r ≥ M" is satisfied at each call.

Back Tracking Algorithm for

Sort W(i) in increasing order.

Bound function is: B(X(1),…, X(k))=true

iff $\sum_{i=1}^{k} W(i)X(i)+\sum_{i=k+1}^{n} W(i)\ge M$ and

$\sum_{i=1}^{k} W(i)X(i)+W(k+1)\le M$

SUMO

SUMOFSUB(*s,k,r*)
X(k)←1
if s+W(k)=M
then print(X(j),j←1 to k)
else if s+W(k)+W(k+1) ≤ M
then call SUMOFSUB(s+W(k),k+1,r-W(k))
if s+r-W(k) ≥ M and s+W(k+1) ≤ M
then X(k)←0
call SUMOFSUB(s,k+1,r-W(k))

Check right son(X(k)=0).

Back Tracking Algorithm for

Sort W(i) in increasing order.

Bound function is: B(X(1),…, X(k))=true

iff $\sum_{i=1}^{k} W(i)X(i)+\sum_{i=k+1}^{n} W(i)\ge M$ and

$\sum_{i=1}^{k} W(i)X(i)+W(k+1)\le M$

SUMO

SUMOFSUB(*s,k,r*)
X(k)←1
if s+W(k)=M
then print(X(j),j←1 to k)
else if s+W(k)+W(k+1) ≤ M
then call SUMOFSUB(s+W(k),k+1,r-W(k))
if s+r-W(k) ≥ M and s+W(k+1) ≤ M
then X(k)←0
call SUMOFSUB(s,k+1,r-W(k))

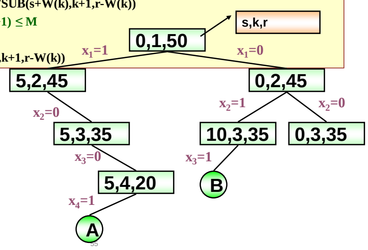
? why not use "k>n" to stop recursion

At each call, s<M, and s+r≥M, so r > 0, it is impossible that k be greater than n。

Case study

```
SUMOFSUB(s,k,r)
  X(k)←1
  if s+W(k)=M
     then  print(X(j),j←1 to k)
     else  if s+W(k)+W(k+1) ≤ M
              then call SUMOFSUB(s+W(k),k+1,r-W(k))
  if s+r-W(k) ≥ M and s+W(k+1) ≤ M
      then  X(k)←0
            call SUMOFSUB(s,k+1,r-W(k))
```

$(w_1, w_2, w_3, w_4) = (5, 10, 15, 20)$;
$M = 25$



(1,0,0,1)

(0,1,1,0)

Simple Review

- N Queen problem
  - Algorithm
  - Bound function
- Subset-sum problem
  - The form of solution, State space tree
  - Bound function
  - Original recursive algorithm
  - Improved recursive algorithm

Design and Analysis of Algorithms

Back-Tracking Algorithms Part III

Design and Analysis of Algorithms

Back-Tracking Algorithms

Topics:

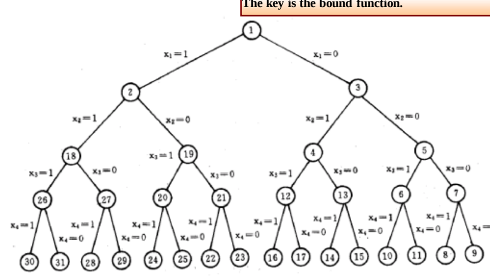
•0/1 Knapsack problem

0-1 knapsack problem

- 0-1 knapsack problem
  - 0-1 knapsack problem: A thief robs a store and finds n items, with item i being worth $\$p_i$ and having weight w_i pounds, The thief can carry at most $M \in$ N in his knapsack but he wants to take as valuable a load as possible. Which item should he take?
  - i.e., $\sum_{1\le i\le n} p_i x_i$ is maximized and s.t. $\sum_{1\le i\le n} w_i x_i \le M$
- The form of solution
  - A solution is defined as an n-tuple $(x_1, x_2, \ldots, x_n)$, where $x_i \in \{0, 1\}$, $1 \le i \le n$. If item i is taken, then $x_i = 1$, otherwise $x_i = 0$.
- An optimization problem : we will find a best answer rather than feasible answer.

State space tree for 0-1 knapsack problem

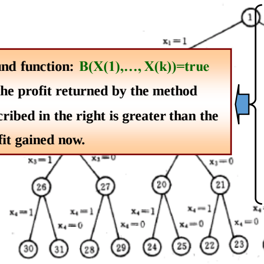
It is easy to determine X(i)/generate problem state.
The key is the bound function.



Bound function

Bound function: B(X(1),…, X(k))=true

Iff the profit returned by the method described in the right is greater than the profit gained now.



Principle: bound function should be helpful to kill some alive nodes.
Our goal is to get max profit, so the bound function should kill those who can not generate greater profit.

up-bound of profit on node Z:
Items have been sorted in decreasing order on P(i)/W(i).
On node Z, we have determined the value of X(i), 1≤i ≤k; then for k+1≤i ≤n, relax X(i)=0 or 1 to 0≤X(i) ≤1, use greedy paradigm to solve the relaxed sub-problem.

Bound function

the value of X(i), 1≤i ≤k have been determined.
p: profit gained before call
w: weight used before call
k: item being checked just now
M: knapsack capacity
return the up-bound profit value (items are sorted in decreasing order on P(i)/W(i)).

```
BOUND( p,w,k,M )
  b←p; c←w
  for i←k+1 to n do
    c←c+W(i)
    if c<M then b←b+P(i)
      else return (b+(1-(c-M)/W(i))*P(i))
  return (b)
```


Bound function

BOUND

```
BOUND( p,w,k,M )
  b←p; c←w
  for i←k+1 to n do
    c←c+W(i)
    if c<M then b←b+P(i)
      else return (b+(1-(c-M)/W(i))*P(i))
  return (b)
```

b: profit gained
c: weight used

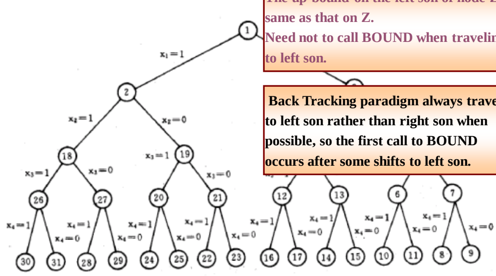
Bound function

BOUND

Determine up-bound using greedy algorithm: check item k+1~n one by one, place all the item into the knapsack if possible, otherwise place part of it into.

```
BOUND( p,w,k,M )
  b←p; c←w
  for i←k+1 to n do
    c←c+W(i)
    if c<M then b←b+P(i)
      else return (b+(1-(c-M)/W(i))*P(i))
  return (b)
```


Bound function



The up-bound on the left son of node Z is same as that on Z.
Need not to call BOUND when traveling to left son.

Back Tracking paradigm always travels to left son rather than right son when possible, so the first call to BOUND occurs after some shifts to left son.

Back Tracking Algorithm for 0-1 knapsack problem

BKNAP1

```
BKNAP1( M,n,W,P,fw,fp,X )
  cw←cp←0;k←1;fp←-1
  loop
    while k≤ n and cw+W(k) ≤ M do
          cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
    if k>n then fp←cp;fw←cw;k←n;X←Y
          else Y(k)←0
    while BOUND(cp,cw,k,M) ≤ fp do
          while k≠0 and Y(k)≠1 do
              k←k-1
          if k=0 then return
          Y(k)←0;cw←cw-W(k);cp←cp-P(k)
  k←k+1
```

Back Tracking Algorithm for 0-1 knapsack problem

BKNAP1

```
BKNAP1( M,n,W,P,fw,fp,X )
   cw←cp←0;k←1;fp←-1
   loop
      while k≤ n and cw+W(k) ≤ M do
            cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
      if k>n then fp←cp;fw←cw;k←n;X←Y
            else Y(k)←0
      while BOUND(cp,cw,k,M) ≤ fp do
            while k≠0 and Y(k)≠1 do
               k←k-1
            if k=0 then return
            Y(k)←0;cw←cw-W(k);cp←cp-P(k)
     k←k+1
```

M: knapsack capacity
n items, weight and profit of each item are stored in W(1:n) and P(1:n), P(i)/W(i)≥P(i+1)/W(i+1);
fw: weight used eventually
fp: profit gained eventually
X: answer vector

67

Back Tracking Algorithm for 0-1 knapsack problem

BKNAP1

```
BKNAP1( M,n,W,P,fw,fp,X )
   cw←cp←0;k←1;fp←-1
   loop
      while k≤ n and cw+W(k) ≤ M do
            cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
      if k>n then fp←cp;fw←cw;k←n;X←Y
            else Y(k)←0
      while BOUND(cp,cw,k,M) ≤ fp do
            while k≠0 and Y(k)≠1 do
               k←k-1
            if k=0 then return
            Y(k)←0;cw←cw-W(k);cp←cp-P(k)
     k←k+1
```

cw: weight used
cp: profit gained

when fp≠-1, X is the vector that gains profit fp
fw,fp,X: information about the best answer currently found.
cw,cp,Y: corresponding information used when searching.

68

Back Tracking Algorithm for 0-1 knapsack problem

BKNAP1

```
BKNAP1( M,n,W,P,fw,fp,X )
   cw←cp←0;k←1;fp←-1
   loop
      while k≤ n and cw+W(k) ≤ M do
            cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
      if k>n then fp←cp;fw←cw;k←n;X←Y
            else Y(k)←0
      while BOUND(cp,cw,k,M) ≤ fp do
            while k≠0 and Y(k)≠1 do
               k←k-1
            if k=0 then return
            Y(k)←0;cw←cw-W(k);cp←cp-P(k)
     k←k+1
```

Travel to left son when possible, store the path in Y.

NOTE: BOUND is not used here.

69

Back Tracking Algorithm for 0-1 knapsack problem

BKNAP1

```
BKNAP1( M,n,W,P,fw,fp,X )
   cw←cp←0;k←1;fp←-1
   loop
      while k≤ n and cw+W(k) ≤ M do
            cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
      if k>n then fp←cp;fw←cw;k←n;X←Y
            else Y(k)←0
      while BOUND(cp,cw,k,M) ≤ fp do
            while k≠0 and Y(k)≠1 do
               k←k-1
            if k=0 then return
            Y(k)←0;cw←cw-W(k);cp←cp-P(k)
     k←k+1
```

if k>n, we get that cp>fp(if cp≤fp, the last call to BOUND will stop the traveling to this leaf), save the answer info.

if k≤n, item k can not be placed into the knapsack, so we must travel to right son(assign 0 to Y(k)).

70

Back Tracking Algorithm for 0-1 knapsack problem

BKNAP1

```
BKNAP1( M,n,W,P,fw,fp,X )
   cw←cp←0;k←1;fp←-1
   loop
      while k≤ n and cw+W(k) ≤ M do
            cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
      if k>n then fp←cp;fw←cw;k←n;X←Y
            else Y(k)←0
      while BOUND(cp,cw,k,M) ≤ fp do
            while k≠0 and Y(k)≠1 do
               k←k-1
            if k=0 then return
            Y(k)←0;cw←cw-W(k);cp←cp-P(k)
     k←k+1
```

Use BOUND to check that whether a better answer is possible, Track back if not.

Track back to the nearest node whose right son has not been generated, generate his right son and check by BOUND.

after while ends, we find a better direction, or the procedure terminates.

71

Back Tracking Algorithm for 0-1 knapsack problem

BKNAP1

```
BKNAP1( M,n,W,P,fw,fp,X )
   cw←cp←0;k←1;fp←-1
   loop
      while k≤ n and cw+W(k) ≤ M do
            cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
      if k>n then fp←cp;fw←cw;k←n;X←Y
            else Y(k)←0
      while BOUND(cp,cw,k,M) ≤ fp do
            while k≠0 and Y(k)≠1 do
               k←k-1
            if k=0 then return
            Y(k)←0;cw←cw-W(k);cp←cp-P(k)
     k←k+1
```

Bound function becomes more and more tighten when searching, for fp becomes greater.

72

Back Tracking Algorithm for 0-1 knapsack problem

BKNAP1

```
BKNAP1( M,n,W,P,fw,fp,X )
  cw←cp←0;k←1;fp←-1
  loop
    while k≤ n and cw+W(k) ≤ M do
          cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
    if k>n then fp←cp;fw←cw;k←n;X←Y
          else Y(k)←0
    while BOUND(cp,cw,k,M) ≤ fp do
          while k≠0 and Y(k)≠1 do
             k←k-1
          if k=0 then return
          Y(k)←0;cw←cw-W(k);cp←cp-P(k)
  k←k+1
```

73

Case Study

M=110,n=8

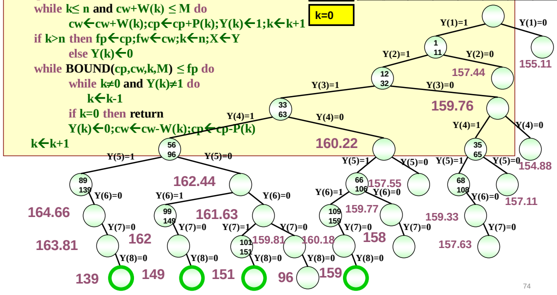
	1	2	3	4	5	6	7	8
P	11	21	31	33	43	53	55	65
W	1	11	21	23	33	43	45	55

```
BKNAP1( M,n,W,P,fw,fp,X )
  cw←cp←0;k←1;fp←-1
  loop
    while k≤ n and cw+W(k) ≤ M do
          cw←cw+W(k);cp←cp+P(k);Y(k)←1;k←k+1
    if k>n then fp←cp;fw←cw;k←n;X←Y
          else Y(k)←0
    while BOUND(cp,cw,k,M) ≤ fp do
          while k≠0 and Y(k)≠1 do
             k←k-1
          if k=0 then return
          Y(k)←0;cw←cw-W(k);cp←cp-P(k)
  k←k+1
```

fw=109
fp=159

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)
1	1	1	0	1	1	0	0

k=0



74

Case Study

M=110,n=8

	1	2	3	4	5	6	7	8
P	11	21	31	33	43	53	55	65
W	1	11	21	23	33	43	45	55

generate 35 of $2^9-1=511$ nodes of the state space tree

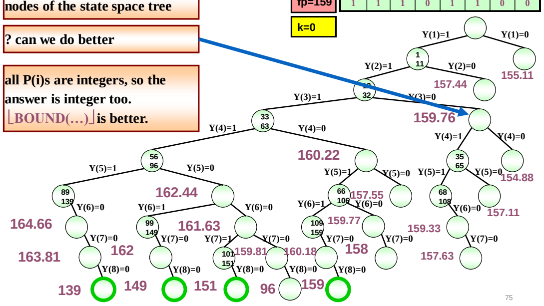
? can we do better

all P(i)s are integers, so the answer is integer too. $\lfloor$BOUND(…)$\rfloor$ is better.

fw=109
fp=159

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)
1	1	1	0	1	1	0	0

k=0



75

Homework

- Solve the following 0-1 knapsack problem use the algorithm discussed today, try to draw the tree generated.
  - n=3, M=22, $(p_1,p_2,p_3)=(30,18,17)$,
  - $(w_1,w_2,w_3)=(15,10,10)$

76

Thanks!