GENERATING COMBINATORIAL OBJECTS (CONT'D)

2.2 Generating k-subsets: Minimal Change Ordering

The minimum Hamming distance possible between k-subsets is 2.

Revolving Door Ordering

It is based on Pascal's Identity: $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$.
We define the sequence of k-subsets $A^{n,k}$ based on $A^{n-1,k}$ and the reverse of $A^{n-1,k-1}$, as follows:

$$\begin{aligned} A^{n,k} &= \left[A_0^{n-1,k}, \ldots, A_{\binom{n-1}{k}-1}^{n-1,k}, |A_{\binom{n-1}{k-1}-1}^{n-1,k-1} \cup \{n\}, \ldots, A_0^{n-1,k-1} \cup \{n\}\right], \\ & \quad for\ 1 \leq k \leq n-1 \\ A^{n,0} &= [\emptyset] \\ A^{n,n} &= [\{1, 2, \ldots, n\}] \end{aligned}$$

Example: Bulding $A^{5,3}$ using $A^{4,3}$ and $A^{4,2}$:

$$\begin{aligned} A^{4,3} &= [\{1,2,3\}, \{1,3,4\}, \{2,3,4\}, \{1,2,4\}] \\ A^{4,2} &= [\{1,2\}, \{2,3\}, \{1,3\}, \{3,4\}, \{2,4\}, \{1,4\}] \end{aligned}$$

$$\begin{aligned} A^{5,3} &= [\{1,2,3\}, \{1,3,4\}, \{2,3,4\}, \{1,2,4\}, | \\ & \quad |\{1,4,\mathbf{5}\}, \{2,4,\mathbf{5}\}, \{3,4,\mathbf{5}\}, \{1,3,\mathbf{5}\}, \{2,3,\mathbf{5}\}, \{1,2,\mathbf{5}\}] \end{aligned}$$

To see that the revolving door ordering is a minimal change ordering, prove:

1. $A^{n,k}_{\binom{n}{k}-1} = \{1, 2, \ldots, k-1, n\}$.
2. $A^{n,k}_0 = \{1, 2, \ldots, k\}$.
3. For any n, k, $1 \leq k \leq n$, $A^{n,k}$ is a minimal ordering of $\mathcal{S}^n_k$.

Ranking

The ranking algorithm is based on the following fact (prove it as an exercise):

$$rank(T) = \sum_{i=1}^{k} (-1)^{k-i} \left(\binom{t_i}{i} - 1 \right) = \begin{cases} \sum_{i=1}^{k} (-1)^{k-i} \binom{t_i}{i}, & k \; even \\ \left[\sum_{i=1}^{k} (-1)^{k-i} \binom{t_i}{i} \right] - 1, & k \; odd \end{cases}$$

Hint: Prove the first equality by induction and the second, directly.

KsubsetRevDoorRank($\vec{T}, k$)

```
r ← −(k mod 2);
s ← 1;
for i ← k downto 1 do
    r ← r + s·C(t_i, i)
    s ← −s;
return r;
```

Unranking

IDEA

Example: $n = 7$, $k = 4$, $r = 8$

$4 \in T, 5, 6, 7 \notin T$	$5 \in T, 6, 7 \notin T$	$6 \in T, 7 \notin T$	$7 \in T$
$\binom{4}{4} = 1$	$\binom{5}{4} = 5$	$\binom{6}{4} = 15$	$\binom{7}{4} = 21$

We can determine the largest element in the set:
$r = 8$ implies $\{\_, \_, \_, 6\}$.

Now, solve it recursively for $n' = 5$, $k' = 3$, $r' = \binom{6}{4} - r - 1 = 6$.

KsubsetRevDoorUnrank(r, k, n)

$x \leftarrow n$;
for $i \leftarrow k$ downto 1 do
  While $\binom{x}{i} > r$ do $x \leftarrow x - 1$;
  $t_i \leftarrow x + 1$
  $r \leftarrow \binom{x+1}{i} - r - 1$;
return $\vec{T}$;

Successor

Let $\vec{T} = [1, 2, 3, \ldots, j-1, t_j, \ldots]$, where $j = \min\{i : t_i \neq i\}$.
Consider fours cases for computing successor:

- Case A: $k \equiv j \pmod 2$
  - Case A1: if $t_{j+1} = t_j + 1$ then move j to the right, and remove $t_j + 1$.
    Example: SUCCESSOR$(\{\underline{1,2,3}, \mathbf{7}, \overline{8}, 12\}) = \{\{\underline{1,2,3,4}, \mathbf{7}, 12\}$.
  - Case A2: if $t_{j+1} \neq t_j + 1$ then move j to the left, and add $t_j + 1$.
    Example:
    SUCCESSOR$(\{\underline{1,2,3}, \mathbf{7}, 10, 12\}) = \{\underline{1,2}, \mathbf{7}, \overline{8}, 10, 12\}$.
- Case B: $k \not\equiv j \pmod 2$
  - Case B1: if $j > 1$ then increment t_{j-1} and (if exists) t_{j-2}.
    Example: SUCCESSOR$(\{\underline{1,2,3}, \mathbf{7}, 10\}) = \{1, \underline{3,4}, \mathbf{7}, 10\}$.
  - Case B2: if $j = 1$ then decrement t_1
    Example: SUCCESSOR$\{7, 9, 10, 12\}) = \{6, 9, 10, 12\}$.

For each case, prove $\textsc{Rank}(\textsc{successor}(T))-\textsc{Rank}(T)=1$.
Case A1: $\textsc{Successor}(\{\underline{1,2,3},\mathbf{7},\overline{8},12\})=\{\{\underline{1,2,3,4},\mathbf{7},12\}$.
$\textsc{rank}(\textsc{successor}(T)-\textsc{rank}(T))=$

$$\begin{aligned}
&= (-1)^{k-j}\binom{t_j+1}{j}+(-1)^{k-j-1}\binom{t_j}{j-1}\\
&\quad -(-1)^{k-j}\binom{t_j}{j}-(-1)^{k-j-1}\binom{j-1}{j-1}\\
&= \left(\binom{t_j+1}{j}-\binom{t_j}{j-1}-\binom{t_j}{j}\right)+\binom{j-1}{j-1}=0+1=1.
\end{aligned}$$

Prove cases A2, B1, B2...

$\textsc{KsubsetRevDoorSuccessor}(\vec{T},k,n)$

$t_{k+1}\leftarrow n+1;$
$j\leftarrow 1;$
While $(j\leq k)$ and $(t_j=j)$ do $j\leftarrow j+1;$
if $(k\not\equiv j \pmod 2)$ then
  if $(j=1)$ then $t_1\leftarrow t_1+1;$ (Case B2)
  else (Case B1)
    $t_{j-1}\leftarrow j;$
    $t_{j-2}\leftarrow j-1;$
else
  if $(t_{j+1}\neq t_j+1)$ then (Case A2)
    $t_{j-1}\leftarrow t_j;$
    $t_j\leftarrow t_j+1$
  else (Case A1)
    $t_{j+1}\leftarrow t_j;$
    $t_j\leftarrow j;$
return $\vec{T}$;

3. Generating Permutations

A permutation is a bijection $\Pi : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$.

We represent it by a list: $\Pi = [\Pi[1], \Pi[2], \ldots, \Pi[n]]$.

3.1. Generating Permutations:Lexicographical Ordering

$n = 3$

rank	permutation
0	$[1, 2, 3]$
1	$[1, 3, 2]$
2	$[2, 1, 3]$
3	$[2, 3, 1]$
4	$[3, 1, 2]$
5	$[3, 2, 1]$

Successor

Example: $\Pi = [3, 5, \mathbf{4}, \underline{7, \overline{6}, 2, 1}]$

Let i = index right before a decreasing suffix = 3.

Let j= index of the successor of $\pi[i] = 4$ in

$\{\Pi[i+1], \ldots, \Pi[n]\} = \{7, 6, 2, 1\}$, $j = 5$.

Swap $\Pi[i]$ and $\Pi[j]$, and reverse $\{\Pi[i+1], \ldots, \Pi[n]\}$.

$$\textsc{Successor}(\Pi) = [3, 5, \mathbf{6}, \underline{1, 2, 4, 7}]$$

Note that

$$\begin{aligned} i &= \max\{l : \Pi[l] < \Pi[l+1]\} \\ j &= \max\{l : \Pi[l] > \Pi[i]\}. \end{aligned}$$

For the algorithm, we add: $\Pi[0] = 0$.

$\textsc{PermLexSuccessor}(n, \Pi)$

```
Π[0] ← 0;
i ← n − 1;
while (Π[i] > Π[i + 1]) do  i ← i − 1;
if (i = 0) then return Π = [1, 2, . . . , n]
j ← n;
while (Π[j] < Π[i]) do  j ← j − 1;
t ← Π[j]; Π[j] ← Π[i]; Π[i] ← t; (swap Π[i] and Π[j])
// In-place reversal of Π[i + 1], . . . , Π[n]:
for h ← i + 1 to ⌊(n−i)/2⌋ do
    t ← Π[h]; Π[h] ← Π[n + i + 1 − h];
    Π[n + i + 1 − h] ← t;
return Π;
```

Ranking

How many permutations come before
$\Pi = [3, 5, 1, 2, 4]$?

the ones of the form $\Pi = [1, \ldots]$ (there are $(n-1)! = 24$ of them)
the ones of the form $\Pi = [2, \ldots]$ (there are $(n-1)! = 24$ of them)

plus the rank of $[5, 1, 2, 4]$ as a permutation of $\{1, 2, 4, 5\}$, which is the standard rank of $[4, 1, 2, 3]$.
So,
$\textsc{Rank}([3, 5, 1, 2, 4]) = 2 \times 4! + \textsc{Rank}([4, 1, 2, 3])$
$= 2 \times 4! + 3 \times 3! + \textsc{Rank}([1, 2, 3])$
$= 2 \times 4! + 3 \times 3! + 0 \times 2! + \textsc{Rank}([1, 2])$
$= 2 \times 4! + 3 \times 3! + 0 \times 2! + 0 \times 1! + \textsc{Rank}([1])$
$= 2 \times 4! + 3 \times 3! + 0 \times 2! + 0 \times 1! + 0 = 66$

General Formula:

$\textsc{Rank}([1], 1) = 0$,
$\textsc{Rank}(\Pi, n) = (\Pi[1] - 1) \times (n-1)! + \textsc{Rank}(\Pi', n-1)$, where

$$\Pi'[i] = \begin{cases} \Pi[i+1] - 1, & \text{if } \Pi[i+1] > \Pi[1] \\ \Pi[i+1], & \text{if } \Pi[i+1] < \Pi[1] \end{cases}$$

PermLexRank(n, Π)

$r \leftarrow 0$;
$\Pi' \leftarrow \Pi$;
for $j \leftarrow 1$ to n do
    $r \leftarrow r + (\Pi'[j] - 1) * (n - j)!$
    for $i \leftarrow j + 1$ to n do
        if $(\Pi'[i] > \Pi'[j])$ then $\Pi'[i] = \Pi'[i] - 1$;
return r;

Unranking

Unranking uses the factorial representation of r.
Let $0 \leq r \leq n! - 1$. Then, $(d_{n-1}, d_{n-2}, \ldots, d_1)$ is the factorial representation of r if

$$r = \Sigma_{i=1}^{n-1} d_i \times i!, \text{ where } 0 \leq d_i < i.$$

(Exercise: prove that such r has a unique factorial representation.)

Examples:

1. $\textsc{Unrank}(15, 4) = [3, 2, 4, 1]$
   $15 = \mathbf{2} \times 3! + \mathbf{1} \times 2! + \mathbf{1} \times 1!$, put $d_0 = \mathbf{0}$.

2	**1**	**1**	**0**
3	2	2	1

2	**1**	**1**	**0**
3	2	2	1

2	**1**	**1**	**0**
3	2	3	1

2	**1**	**1**	**0**
3	2	4	1

2. $\textsc{Unrank}(8, 4) = [2, 3, 1, 4]$
   $8 = \mathbf{1} \times 3! + \mathbf{1} \times 2! + \mathbf{0} \times 1!$,

1	**1**	**0**	**0**
2	2	1	1

1	**1**	**0**	**0**
2	2	1	2

1	**1**	**0**	**0**
2	2	1	3

1	**1**	**0**	**0**
2	3	1	4

3. $\textsc{Unrank}(21, 4) = [4, 2, 3, 1]$
   $21 = \mathbf{3} \times 3! + \mathbf{1} \times 2! + \mathbf{1} \times 1!$,

3	**1**	**1**	**0**
4	2	2	1

3	**1**	**1**	**0**
4	2	2	1

3	**1**	**1**	**0**
4	2	3	1

3	**1**	**1**	**0**
4	2	3	1

Justification: $\Pi[1] = d_{n-1} + 1$ because exactly d_{n-1} blocks of size $(n-1)!$ come before Π.
$\Pi[2], \Pi[3], \ldots, \Pi[n]$ is computed from permutation Π', in the following way:

$r' = r - d_{n-1} \times (n-1)!$
$\Pi' =$UNRANK$(r', n-1)$,

$$\Pi[i] = \begin{cases} \Pi'[i-1], & \text{if } \Pi'[i-1] < \Pi[1] \\ \Pi'[i-1]+1, & \text{if } \Pi'[i-1] > \Pi[1] \end{cases} \qquad \text{for } 2 \leq i \leq n$$

PERMLEXUNRANK(r, n)
    $\Pi[n] \leftarrow 1$;
    for $j \leftarrow 1$ to $n-1$ do
        $d \leftarrow \frac{r \bmod (j+1)!}{j!}$; // calculates d_j
        $r \leftarrow r - d * j!$;
        $\Pi[n-j] \leftarrow d + 1$;
        for $i \leftarrow n-j+1$ to n do
            if $(\Pi[i] > d)$ then $\Pi[i] \leftarrow \Pi[i] + 1$;
    return Π;

3.2. Generating permutations:Minimal Change Ordering

Minimal change for permutations: two permutatiosn must differ by adjacent transposition.

The Trotter-Johnson algorithm follows the following ordering:

$$
\begin{aligned}
T^1 &= [[1]] \\
T^2 &= [[1, \mathbf{2}], [\mathbf{2}, 1]] \\
T^3 &= [[1, 2, \mathbf{3}], [1, \mathbf{3}, 2], [\mathbf{3}, 1, 2][\mathbf{3}, 2, 1], [2, \mathbf{3}, 1], [2, 1, \mathbf{3}]]
\end{aligned}
$$

How to build T^3 using T^2:

	1		2	**3**
	1	**3**	2	
3	1		2	
3	2		1	
	2	**3**	1	
	2		1	**3**

See picture for T^4 in page 58 of the textbook.

Ranking

Let

$$\Pi = [\Pi[1], \ldots, \Pi[k-1], \mathbf{\Pi[k] = n}, \Pi[k+1], \ldots, \Pi[n]].$$

Thus, Π is built from Π' by inserting n, where

$$\Pi' = [\Pi[1], \ldots, \Pi[k-1], \Pi[k+1], \ldots, \Pi[n]].$$

$\textsc{Rank}(\Pi, n) = n \times \textsc{Rank}(\Pi', n-1) + E,$

$$E = \begin{cases} n-k, & \text{if Rank}(\Pi', n-1) \text{ is } even \\ k-1, & \text{if Rank}(\Pi', n-1) \text{ is } odd \end{cases}$$

Example:
$\textsc{Rank}([3,4,2,1],4) = 4 \times \textsc{Rank}([3,2,1],3) + E =$
$4 \times 3 + (2-1) = 13.$

```
PermTrotterJohnsonRank(Π, n)
    r ← 0;
    for j ← 2 to n do
        k ← 1; i ← 1;
        while (Π[i] ≠ j) do
            if (Π[i] < j) then k ← k + 1;
            i ← i + 1;
        if (r ≡ 0 mod 2) then r ← j * r + j − k;
                         else r ← j * r + k − 1;
    return r;
```

Unranking

Based on similar recursive principle.
Let $r' = \lfloor \frac{r}{n} \rfloor$, $\Pi' =$UNRANK$(r', n-1)$.
Let $k = r - n \times r'$.
Insert n into Π' in position:

$$\begin{cases} k+1, & \text{if } r' \text{ is odd} \\ n-k, & \text{if } r' \text{ is even} \end{cases}$$

PERMTROTTERJOHNSONUNRANK(n, r)

```
Π[1] ← 1;
r_2 ← 0;
for j ← 2 to n do
    r_1 ← ⌊(r*j!)/n!⌋;    // rank of Π when restricted to {1, 2, ..., j}
    k ← r_1 − j * r_2;
    if (r_2 is even) then
      for i ← j downto j − k do
          Π[i + 1] ← Π[i];
      Π[j − k] ← j;
    else
        for i ← j − 1 downto k + 1 do
            Π[i + 1] ← Π[i];
        Π[k + 1] ← j;
    r_2 ← r_1;
return Π;
```

Successor

There are four cases to analyse:

- RANK(Π') is even
  - If possible, move left:
    SUCCESSOR($[1, \mathbf{4}, 2, 3]$)=($[\mathbf{4}, 1, 2, 3]$)
  - If n is in first position, get successor of the remaining permutation: SUCCESSOR($[\mathbf{4}, 1, 2, 3]$)=($[\mathbf{4}, 1, 3, 2]$),
- RANK(Π') is odd
  - If possible, move right:
    SUCCESSOR($[3, \mathbf{4}, 2, 1]$)=($[3, 2, \mathbf{4}, 1]$)
  - If n is in last position, get successor of the remaining permutation: SUCCESSOR($[3, 2, 1, \mathbf{4}]$)=($[2, 3, 1, \mathbf{4}]$).

We need to be able to determine the parity of RANK(Π').

The parity of a permutation is the parity of the number of interchanges necessary for transforming the permutation into $[1, 2, \ldots, n]$.
$\Pi' = [5, 1, 3, 4, 2]$ is an even permutation since 2 steps are sufficient to convert it into $[1, 2, 3, 4, 5]$.

Note that: parity of RANK(Π') = parity of Π', since in the Trotter-Johnson algorithm $[1, 2, \ldots, n]$ has rank 0, and eac swap increases the rank by 1.

It is easy to compute the parity of a permutation in $\Theta(n^2)$:

$\textsc{PermParity}(n, \Pi) = |\{(i,j) : \Pi[i] > \Pi[j], 1 \leq i \leq j \leq n\}|$.

See the textbook for a $\Theta(n)$ algorithm.

$\textsc{PermTrotterJohnsonSuccessor}(n, \Pi)$

$s \leftarrow 0$;
for $i \leftarrow 1$ to n do $\rho[i] \leftarrow \Pi[i]$;
done $\leftarrow$ false;
m $\leftarrow$ n;
while $(m > 1)$ and (not done) do
  $d \leftarrow 1$;
  while $(\rho[d] \neq m)$ do $d \leftarrow d + 1$;
  for $i \leftarrow d$ to $m - 1$ do $\rho[i] \leftarrow \rho[i + 1]$;
  par $\leftarrow \textsc{PermParity}(m - 1, \rho)$;
  if (par $= 1$) then
    if $(d = m)$ then $m \leftarrow m - 1$;
    else swap $\Pi[s + d], \Pi[s + d + 1]$
      done $\leftarrow$ true;
  else
    if $(d = 1)$ then $m \leftarrow m - 1$; $s \leftarrow s + 1$
    else swap $\Pi[s + d], \Pi[s + d + 1]$
      done $\leftarrow$ true;
if $(m = 1)$ then return $[1, 2, \ldots, n]$
  else return Π;

EXHAUSTIVE GENERATION AND SEARCH: BACKTRACKING

Backtracking Algorithms

Knapsack (Optimization) Problem

Instance: Profits $p_0, p_1, \ldots, p_{n-1}$
Weights $w_0, w_1, \ldots, w_{n-1}$
Knapsack capacity M

Find: and n-tuple $[x_0, x_1, \ldots, x_{n-1}] \in \{0,1\}^n$
such that $P = \sum_{i=0}^{n-1}$ s maximized,
subject to $\sum_{i=0}^{n-1} w_i x_i \leq M$.

Example:

Objects:	1	2	3	4
weight (lb)	8	1	5	4
profit	\$500	\$1,000	\$ 300	\$ 210

Knapsack capacity: $M = 10$ lb.

Two feasible solutions and their profit:

x_1	x_2	x_3	x_4	profit
1	1	0	0	\$ 1,500
0	1	1	1	\$ 1,510

This problem is NP-hard.

Naive Backtracking Algorithm for Knapsack

Examine all 2^n tuples and keep the ones with maximum profit.

Global Variables $X, OptP, OptX$.
Algorithm KNAPSACK1 (l)
    if $(l = n)$ then
        if $\Sigma_{i=0}^{n-1} w_i x_i \leq M$ then
            $CurP \leftarrow \Sigma_{i=0}^{n-1} p_i x_i$;
            if $(CurP > OptP)$ then
                $OptP \leftarrow CurP$;
                $OptX \leftarrow [x_0, x_1, \ldots, x_{n-1}]$;
    else $x_l \leftarrow 1$;
        KNAPSACK1 $(l+1)$;
        $x_l \leftarrow 0$;
        KNAPSACK1 $(l+1)$;

First call: KNAPSACK1 (0).

Running time: 2^n n-tuples are checked, and it takes $\Theta(n)$ to check each solution. The total running time is $\Theta(n2^n)$.

Note: not all n-tuples are feasible but the algorithm will test all (the whole search tree is examined).

We will improve this algorithm!!!

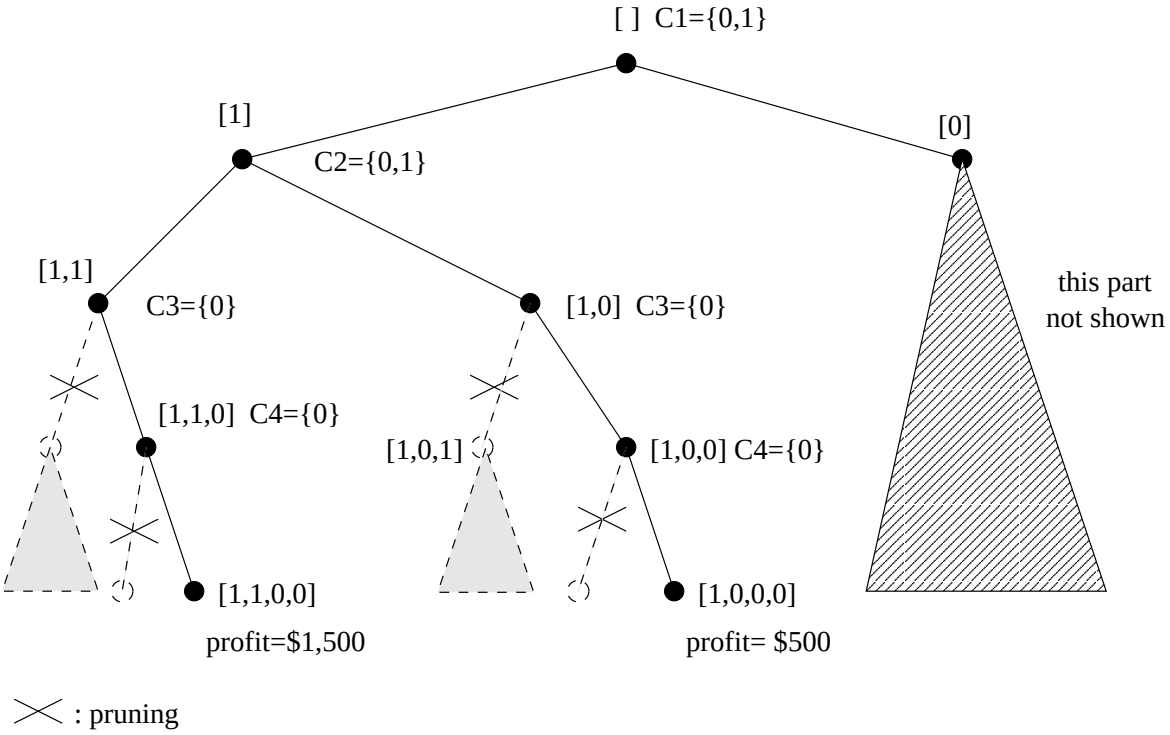
A General Backtracking Algorithm

- Represent a solution as a list: $X = [x_0, x_1, x_2, \ldots]$.
- Each $x_i \in P_i$ (possibility set)
- Given a partial solution: $X = [x_0, x_1, \ldots, x_l]$, we can use constraints of the problem to limit the choice of x_l to $\mathcal{C}_l \subseteq P_l$ (choice set).
- By computing $\mathcal{C}_l$ we prune the search tree, since for all $y \in P_l \setminus \mathcal{C}_l$ the subtree rooted on $[x_0, x_1, \ldots, x_l, y]$ is not considered.

Part of the search tree for the previous Knapsack example:

w_i	8	1	5	4
p_i	\$500	\$1,000	\$ 300	\$ 210

$M = 10$.



General Backtracking Algorithm with Pruning

Global Variables $X = [x_0, x_1, \ldots]$, $\mathcal{C}_l$, for $l = 0, 1, \ldots$).

Algorithm BACKTRACK (l)

if $X = [x_0, x_1, \ldots, x_{l-1}]$ is a feasible solution) then

"Process it"

Compute $\mathcal{C}_l$;

for each $x \in \mathcal{C}_l$ do

$x_l \leftarrow x$;

BACKTRACK$(l + 1)$;

Backtracking with Pruning for Knapsack

Global Variables $X, OptP, OptX$.

Algorithm KNAPSACK2 $(l, CurW)$

```
    if (l = n) then
      if (Σ_{i=0}^{n-1} p_i x_i > OptP) then
        OptP ← (Σ_{i=0}^{n-1} p_i x_i;
        OptX ← [x_0, x_1, ..., x_{n-1}];
    if (l = n) then C_l ← ∅
    else if (CurW + w_l ≤ M) then
           C_l ← {0, 1};
         else C_l ← {0};
    for each x ∈ C_l do
        x_l ← x;
        KNAPSACK2 (l + 1, CurW + w_l x_l);
```

First call: KNAPSACK2 $(0, 0)$.