Dynamic Programming

- Data Structures and Algorithms
- Kyuseok Shim
- SoEECS, SNU.

Dynamic Programming

- It is applicable when the sub-problems are <u>*not*</u> independent -> applies when the subproblems overlab (subproblems share subsubproblems)
- It solves every sub-problem just once and then saves its answer in a table, thereby avoiding re-computing
- In contrast,
  - In Divide-and-Conquer algorithm, sub-problems share sub-problems
  - Divide-and-Conquer algorithm does more work than necessary
  - Divide-and-Conquer algorithm repeatedly solve the common sub-problems

Dynamic Programming

- It is typically applied to optimization problem
  - In optimization problems,
    - There can be many possible solutions
    - Each solution has a value
    - Wish to find a solution with the optimal value
    - There can be several solutions achieving the optimal value
    - We call such a solution an optimal solution to the problem, as opposed to the optimal solution

Dynamic Programming

- Characterize the structure of an optimal solution
- Recursively define the value of an optimal solution
- Compute the value of an optimal solution in a bottom-up fashion
- Construct an optimal solution from computed information

Rod cutting

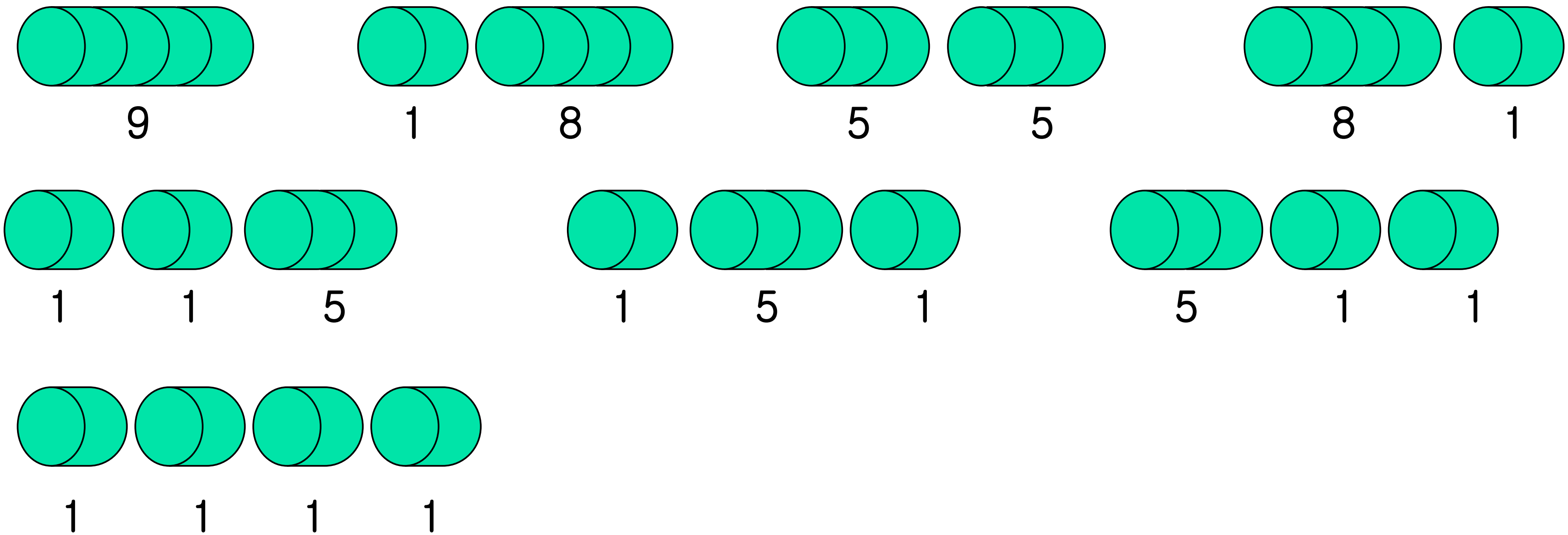
Problem Definition

- Given a rod of length n inches and a table of prices p_i for i=1,2,…,n
- Determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces

Length i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Price p_i	1	5	8	9	10	17	17	20	24	30

- Optimal solution : cuts the rod into k pieces, $1 \le k \le n$
  - $n = i_1 + i_2 + \ldots + i_k$
  - $r_n = p_{i1} + p_{i2} + \ldots + p_{ik}$

Problem Definition



The 8 possible ways of cutting up a rod of length 4.

Example

Length i	1	2	3	4	5	6	7	8	9	10
Price p_i	1	5	8	9	10	17	17	20	24	30

- $r_1=1$ from solution $1=1$ (no cuts)
- $r_2=5$ from solution $2=2$ (no cuts)
- $r_3=8$ from solution $3=3$ (no cuts)
- $r_4=10$ from solution $4=2+2$
- $r_5=13$ from solution $5=2+3$
- $r_6=17$ from solution $6=6$(no cuts)
- $r_7=18$ from solution $7=1+6$ or $2+2+3$
- $r_8=22$ from solution $8=2+6$
- $r_9=25$ from solution $9=3+6$
- $r_{10}=30$ from solution $10=10$ (no cuts)

- $r_n=\max(p_n, r_1+r_{n-1}, r_2+r_{n-2},\ldots,r_{n-1}+r_1)$

Rod cutting

- $r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)$
  $= \max(p_n, \max_{1 \le i \le n}(r_i + r_{n-i}))$
- Rod-cutting problem exhibits optimal substructure:
  - optimal solutions to a problem incorporate optimal solutions to related subproblems, which we may solve independently

- Only the remainder (not the first piece) may be further divided
- $r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$ (first piece : size i and revenue p_i)
- optimal solution embodies the solution to only one related subproblem – the remainder –rather than tow

Rod cutting

- cf) $r_n = \max(p_n, \max_{1 \le i \le n}(r_i + r_{n-i}))$

$$T(n) = 1 + \sum_{j=1}^{n-1} \big(T(j) + T(n-j)\big)$$

$$T(n-1) = 1 + \sum_{j=1}^{n-2} \big(T(j) + T(n-1-j)\big)$$

$$T(n) - T(n-1) = \sum_{j=1}^{n-1} \big(T(j) + T(n-j)\big) - \sum_{j=1}^{n-2} \big(T(j) + T(n-1-j)\big) = 2T(n-1)$$

$$T(n) = 3T(n-1)$$

$$T(n) = T(0) \cdot 3^n = 3^n$$

Recursive top-down implementation

- Cut-Rod(p,n)

```
if n==0
    return 0
q=-∞
for i=1 to n
    q=max(q,p[i]+Cut-Rod(p,n-i))
return q
```

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j), \qquad T(n) = 2^n$$

Recursive top-down implementation

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$

$$T(n-1) = 1 + \sum_{j=0}^{n-2} T(j)$$

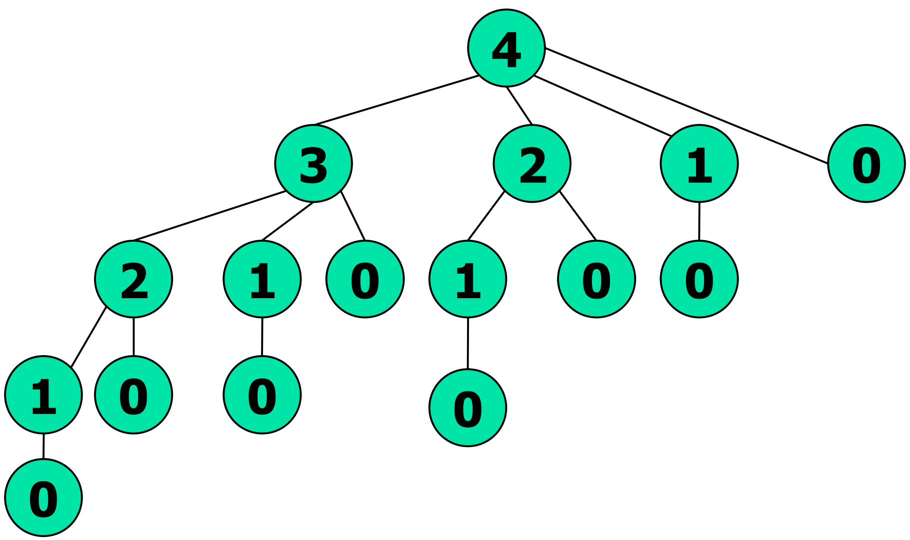
$$T(n) - T(n-1) = \sum_{j=0}^{n-1} T(j) - \sum_{j=0}^{n-2} T(j) = T(n-1)$$

$$T(n) = 2T(n-1)$$

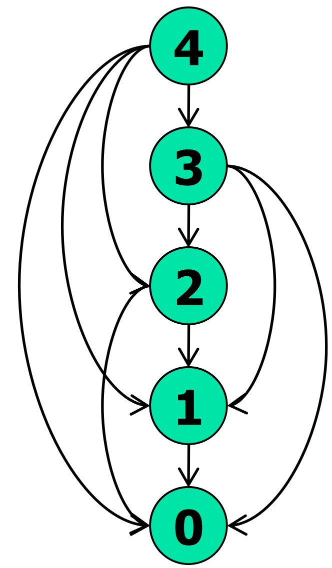
$$T(n) = T(0) \cdot 2^n = 2^n$$

Using dynamic programming for optimal rod cutting

- Naïve recursive solution is inefficient (∵solve the same subproblems repeatedly) -> solve each subproblem once by saving solution



recursion tree of recursive version when n=4
In general, 2^n nodes 2^{n-1} leaves



recursion tree of
dynamic programming

Top-down cut-rod procedure with memoization added

- Memoized-Cut-Rod(p,n)

```
let r[0…n] be a new array
for i=0 to n
        r[i]=-∞
return Memoized-Cut-Rod-Aux(p,n,r)
```

- Memoized-Cut-Rod-Aux(p,n,r)

```
if r[n]≥0
        return r[n]
if n==0
        q=0
else q=-∞
        for i=1 to n
                q=max(q,p[i]+Memoized-Cut-Rod-Aux(p,n-i,r))
r[n]=q
return q
```

Bottom-up version

- Bottom-Up-Cut-Rod(p,n)

```
let r[0…n] be a new array
r[0]=0
for j=1 to n
        q=-∞
        for i=1 to j
                q=max(q,p[i]+r[j-i])
        r[j]=q
return r[n]
```

- $T(n)=\Theta(n^2)$ ∵ doubly-nested loop structure

Reconstructing a solution

- Extend approach to record
  - the optimal value computed for each subproblem
  - a choice that led to the optimal value

Reconstructing a solution

- Extended-Bottom-Up-Cut-Rod(p,n)

```
1.  let r[0…n] and s[0…n] be new arrays
2.  r[0]=0
3.  for j=1 to n
4.      q=-∞
5.      for i=1 to j
6.          if q<p[i]+r[j-i]
7.              q=p[i]+r[j-i]
8.              s[j]=i
9.      r[j]=q
10. return r and s
```

- s[j] in line 8 hold the optimal size i of the first piece to cut off when solving a subproblem of size j

Reconstructing a solution

- Print-Cut-Rod-Solution (p,n)

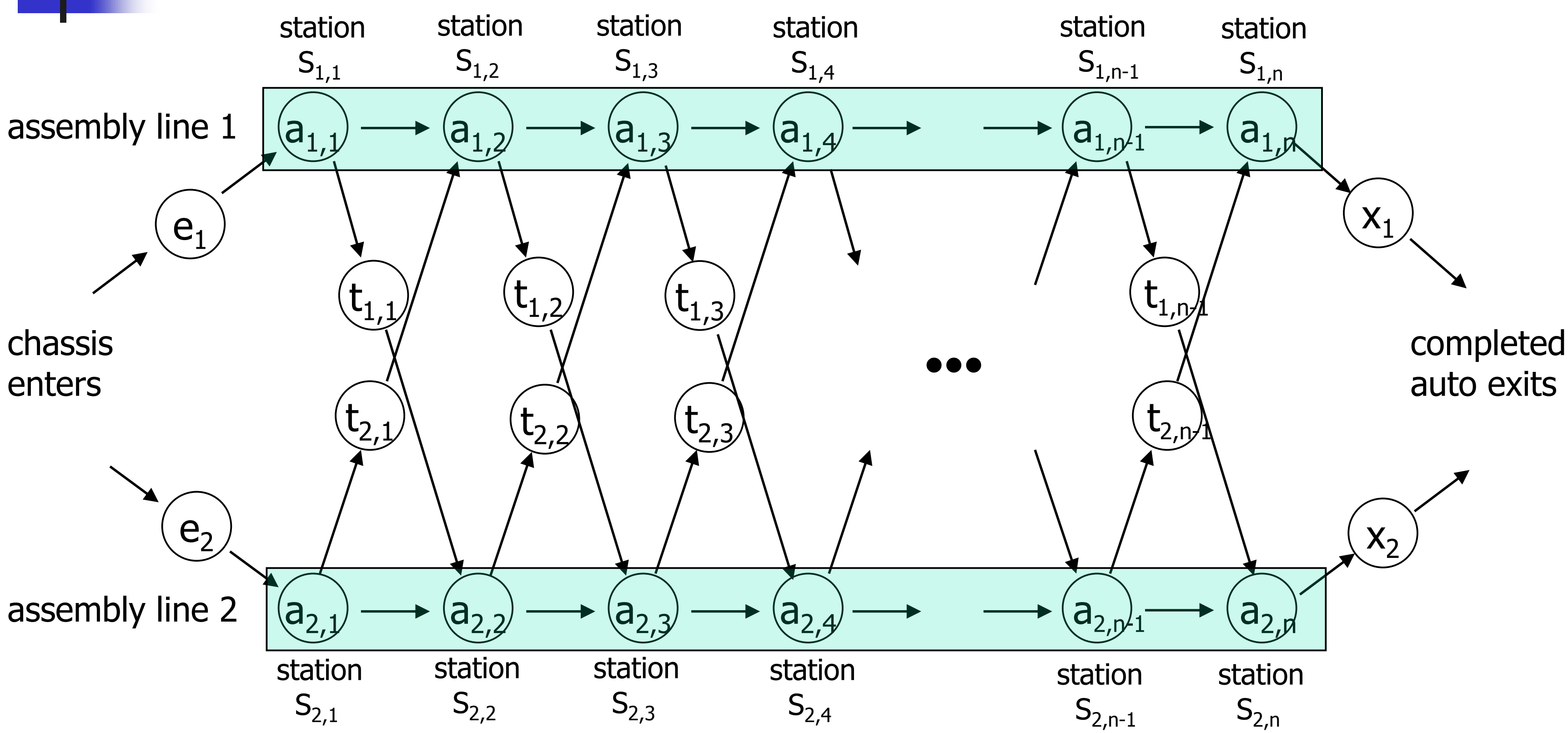
1. (r,s)=Extended-Bottom-Up-Cut-Rod(p,n)
2. while n>0
3. print s[n]
4. n=n-s[n]

i	0	1	2	3	4	5	6	7	8	9	10
r[i]	0	1	5	8	10	13	17	18	22	25	30
s[i]	0	1	2	3	2	2	6	1	2	3	10

- Print-Cut-Rod-Solution (p,7) : print 1 and 6

Assembly-line Scheduling

Assembly-line



Assembly-line

- $S_{i,j}$: station j on line i
- $a_{i,j}$: the assembly time required at station $S_{i,j}$
- $t_{i,j}$: the time to transfer a chassis from assembly line i, after having gone through station $S_{i,j}$
- e_i, x_i : entry time on line i, exit time on line i

Problem Definition

- There are two assembly lines each with n stations
- The j-th station on line 1, 2 performs the same function
- Determine which stations to choose in order to minimize the total time through the factory

Brute Force Way

- There are 2^n possible ways to choose stations
- So brute force way take $\Omega(2^n)$ time complexity
- It is infeasible when n is large

The Structure of the Fastest Way through the Factory (1)

- The fastest way through station $S_{1,j}$ is either
  - the fastest way through station $S_{1,j-1}$ and then directly through station $S_{1,j}$ or
  - the fastest way through station $S_{2,j-1}$, a transfer from line 2 to line 1, and then through station $S_{1,j}$

The Structure of the Fastest Way through the Factory (2)

- The fastest way through station $S_{2,j}$ is either
  - the fastest way through station $S_{2,j-1}$ and then directly through station $S_{2,j}$ or
  - the fastest way through station $S_{1,j-1}$, a transfer from line 1 to line 2, and then through station $S_{2,j}$

The Structure of the Fastest Way through the Factory (3)

- Solving the problem of finding the fastest way through station j of either line
  - need to solve the sub problems of finding the fastest ways through station j-1 on both lines

Recursive Solution

- $f_i[j]$: the fastest time to get a chassis from starting point through station $S_{i,j}$

$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1 \\ \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \ge 2 \end{cases}$$

$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1 \\ \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \ge 2 \end{cases}$$

- The fastest way through the entire factory

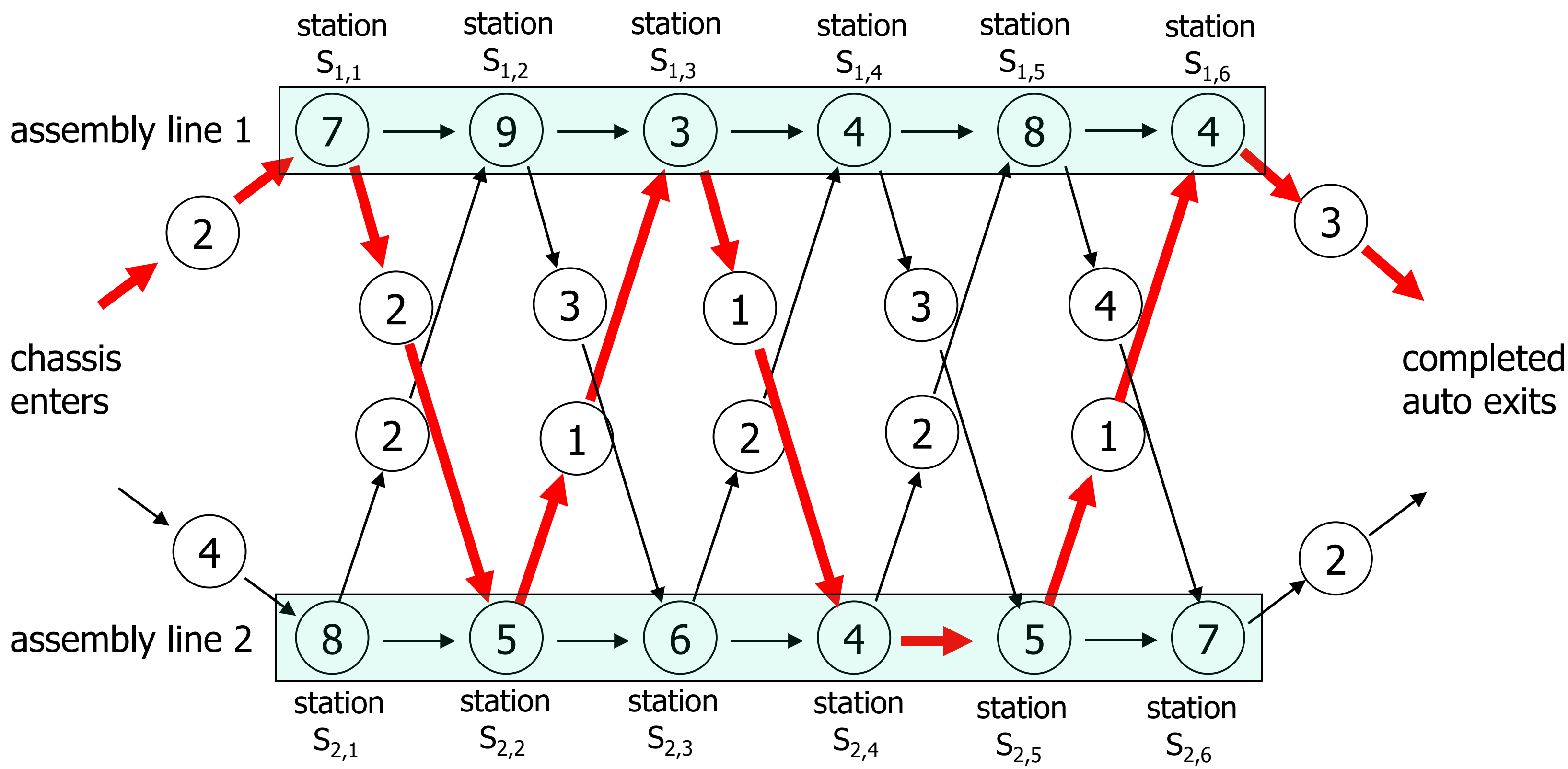
$$f^* = \min(f_1[n] + x_1,\ f_2[n] + x_2)$$

Computing the Fastest Times (1)

- We can write a recursive algorithm based on previous recurrence relation
- Let $r_i(j)$ be # of references made to $f_i[j]$ in a recursive algorithm
  - $r_1(n)=r_2(n)=1$
  - $r_1(j)=r_2(j)=r_1(j+1)+r_2(j+1)$ for $j=1,\ldots,n-1$
  - $r_i(j)=2^{n-j}$ (Exercise 15.1-3)
  - $f_1[1]$ alone is referenced 2^{n-1} times
- Thus, the running time is exponential in n

Computing the Fastest Times (2)

- By computing the $f_i[j]$ values in order of increasing station numbers j
  - Can compute the fastest way through the factory in $\Theta(n)$ time



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$f^* = 38$

$l^* = 1$

FASTEST-WAY(a,t,e,x,n)

$f_1[1] \leftarrow e_1+a_{1,1}$
$f_2[1] \leftarrow e_2+a_{2,1}$
for $j \leftarrow 2$ to n
    do if $f_1[j-1]+a_{1,j} \leq f_2[j-1]+t_{2,j-1}+a_{1,j}$
        then $f_1[j] \leftarrow f_1[j-1]+a_{1,j}$
            $l_1[j] \leftarrow 1$
        else $f_1[j] \leftarrow f_2[j-1]+t_{2,j-1}+a_{1,j}$
            $l_1[j] \leftarrow 2$
    if $f_2[j-1]+a_{2,j} \leq f_1[j-1]+t_{1,j-1}+a_{2,j}$
        then $f_2[j] \leftarrow f_2[j-1]+a_{2,j}$
            $l_2[j] \leftarrow 2$
        else $f_2[j] \leftarrow f_1[j-1]+t_{1,j-1}+a_{2,j}$
            $l_2[j] \leftarrow 1$
if $f_1[n]+x_1 \leq f_2[n]+x_2$
    then $f^* = f_1[n]+x_1$
        $l^* = 1$
    else $f^* = f_2[n]+x_2$
        $l^* = 2$