Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Corso di Algoritmi e Strutture Dati

Programmazione dinamica

Programmazione dinamica

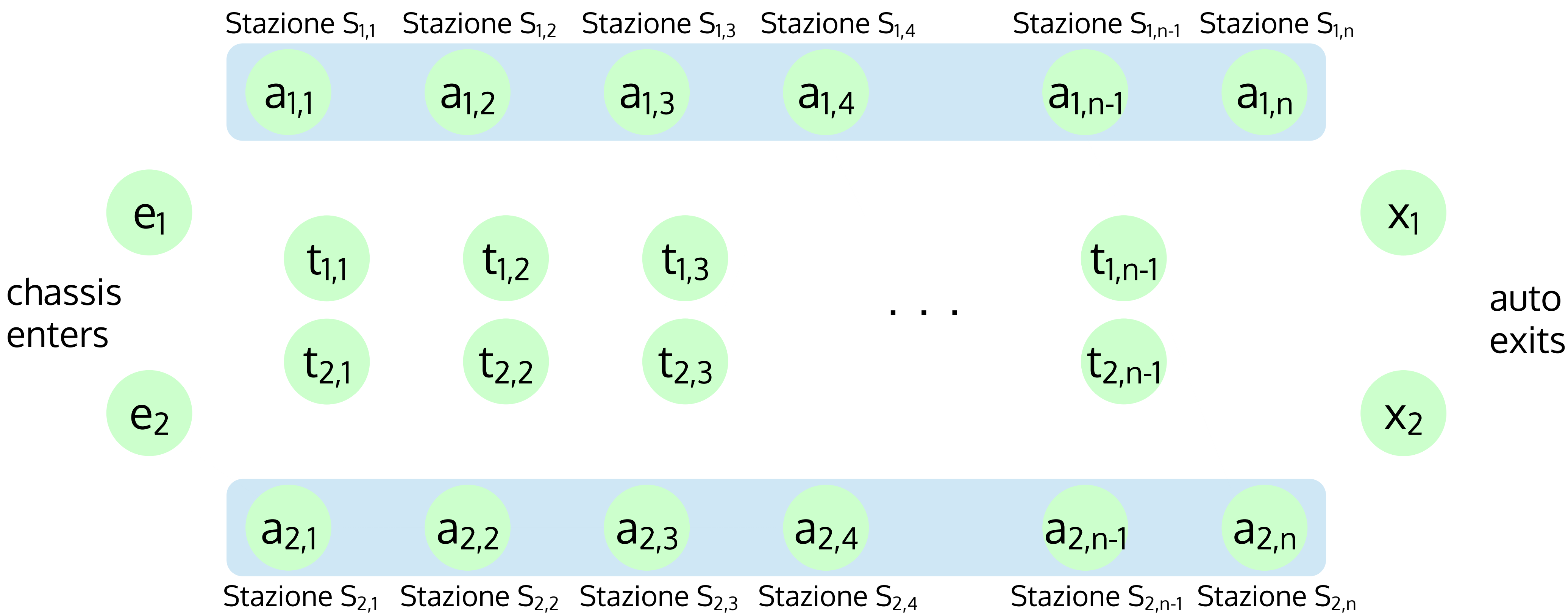
- La programmazione dinamica è un approccio che si può utilizzare in diversi casi per risolvere efficientemente un *problema di ottimizzazione*
  - Diverse soluzioni possibili per un problema
  - Ogni soluzione ha un costo, il problema è trovare *una* soluzione con il costo minimo o massimo
- Come divide-et-impera, la programmazione dinamica risolve un problema combinando le soluzioni di sottoproblemi
- Diversamente da divide-et-impera, la programmazione dinamica si applica anche quando i sottoproblemi non sono indipendenti
- La programmazione dinamica risolve i problemi in comune una sola volta, divide-et-impera più volte

Programmazione dinamica

- Un algoritmo di programmazione dinamica può essere sviluppato seguendo quattro passi
  1. Caratterizzazione della struttura di una soluzione ottima
  2. Definizione ricorsiva del valore di una soluzione ottima
  3. Calcolo del valore di una soluzione ottima in modo bottom-up
  4. Costruzione di una soluzione ottima dalle informazioni calcolate

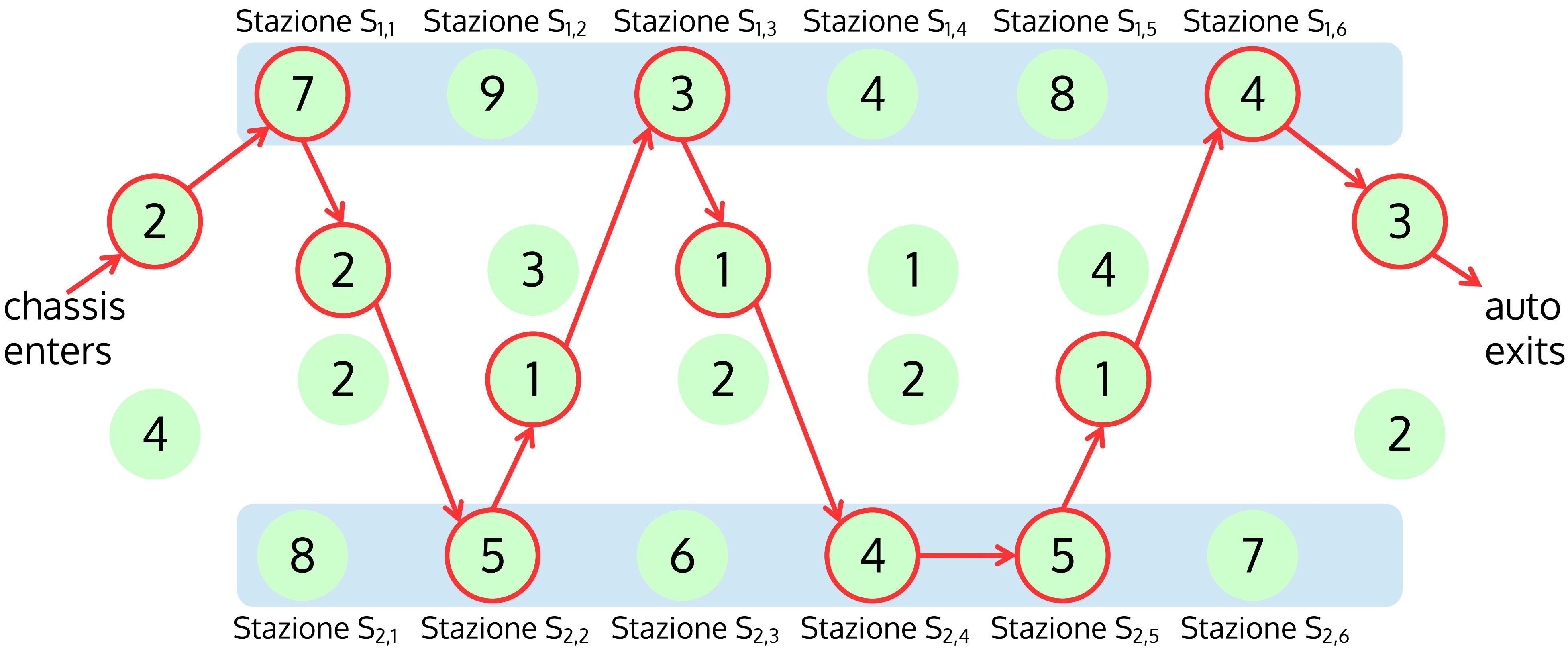
- L'ultimo passo non è necessario se è richiesto solo il valore ottimo e non una soluzione ottima

Catene di montaggio



- Uno chassis deve attraversare n stazioni (su una qualunque delle due linee)
- I tempi di stazionamento sono $a_{i,j}$, quelli di trasferimento $t_{i,j}$
- Determinare un "percorso" che richiede il tempo minimo

Catene di montaggio



- Il percorso più veloce è evidenziato
- Dato il percorso, calcolarne il tempo richiede $\Theta(n)$
- Ma ci sono 2^n possibili percorsi
- L'approccio forza bruta non è fattibile

Struttura di una soluzione ottima

- Consideriamo il percorso più veloce fino alla stazione $S_{1,j}$
  - j>1, altrimenti è banale (si entra nella linea 1)
  - Lo chassis potrebbe provenire:
  - da $S_{1,j-1}$
    - Lo chassis deve aver preso il percorso più veloce fino a $S_{1,j-1}$
  - da $S_{2,j-1}$ e poi trasferito su linea 1
    - Lo chassis deve aver preso il percorso più veloce fino a $S_{2,j-1}$

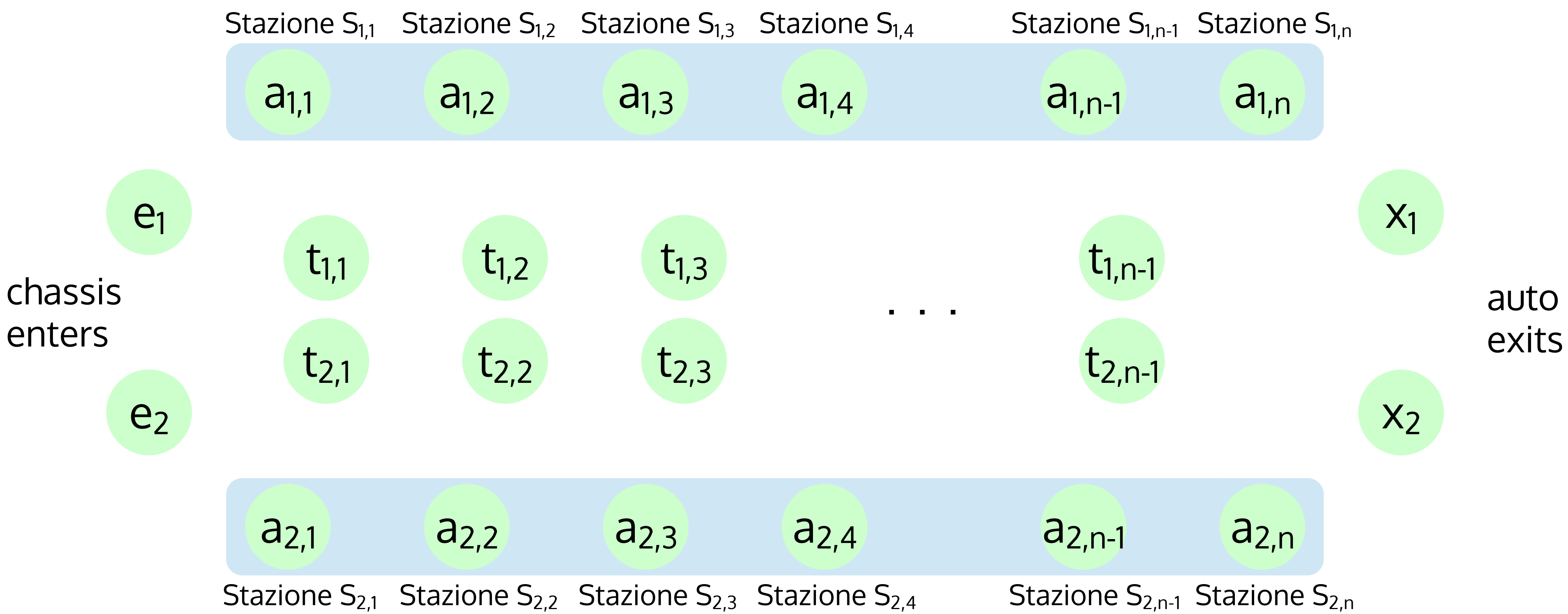
- In ogni caso, *la soluzione ottima di un problema contiene al suo interno la soluzione ottima di un sottoproblema*
- Il percorso più veloce fino a $S_{1,j}$ è dato da
  - Percorso più veloce fino a $S_{1,j-1}$ e poi direttamente a $S_{1,j}$, oppure
  - Percorso più veloce fino a $S_{2,j-1}$ e poi trasferimento a $S_{1,j}$

Definizione ricorsiva del valore ottimo

- Si definisce il valore di una soluzione ottima in maniera ricorsiva in funzione delle soluzioni ottime dei sottoproblemi

- Sia f^* il tempo minimo di completamento e $f_i[j]$ il tempo minimo per arrivare all'uscita della stazione $S_{i,j}$

Definizione ricorsiva del valore ottimo



$f^* = \min(f_1[n]+x_1, f_2[n]+x_2)$; $f_1[1] = e_1+a_{1,1}$; $f_2[1] = e_2+a_{2,1}$

$f_1[j] = \min(f_1[j-1]+a_{1,j}, f_2[j-1]+t_{2,j-1}+a_{1,j})$

$f_2[j] = \min(f_2[j-1]+a_{2,j}, f_1[j-1]+t_{1,j-1}+a_{2,j})$

Definizione ricorsiva del valore ottimo

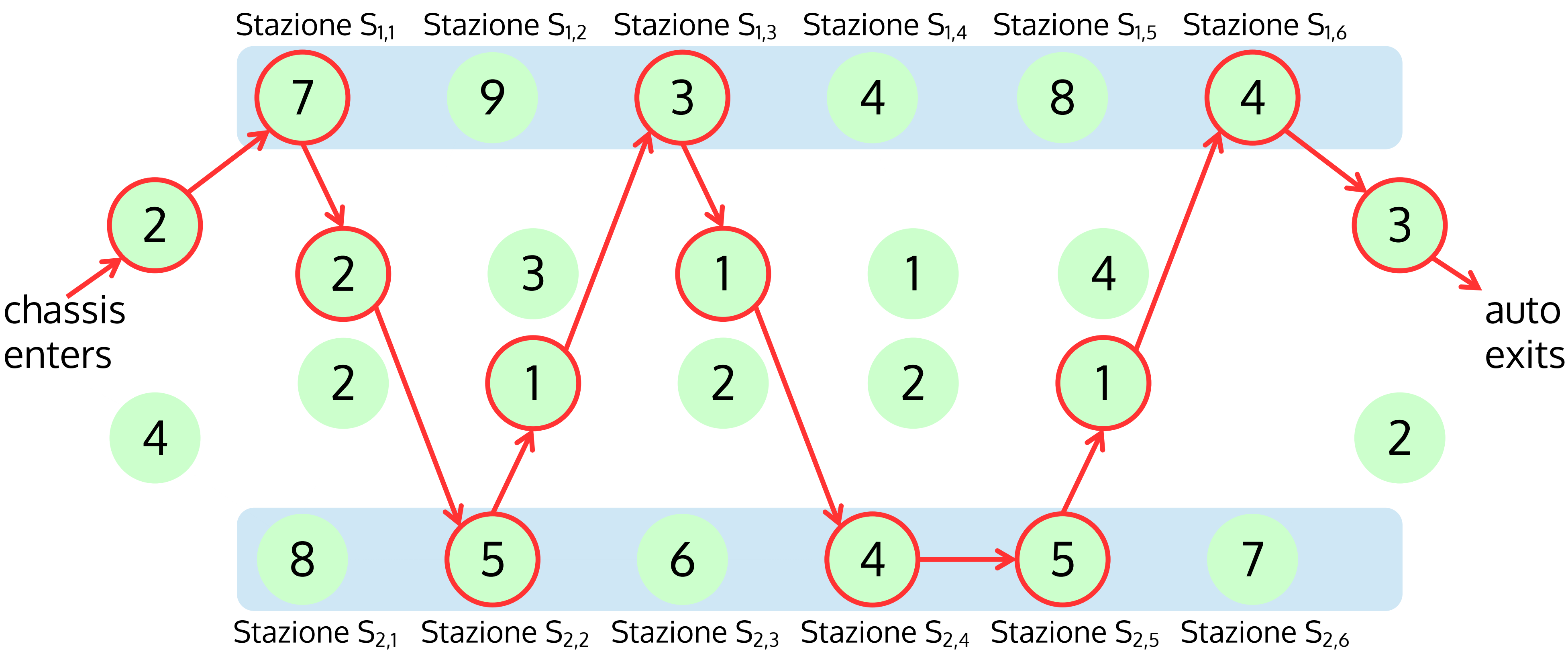
- Si ottengono le seguenti equazioni ricorsive:

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

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

- Per costruire il percorso più veloce, teniamo traccia di:
  - $l_i[j]$ la linea (1 o 2) su cui si trova la stazione (j-1) che precede $S_{i,j}$ lungo il percorso più veloce verso $S_{i,j}$
  - l* la linea su cui si trova la stazione n nel percorso più veloce

Catene di montaggio



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

$f^*=38$

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

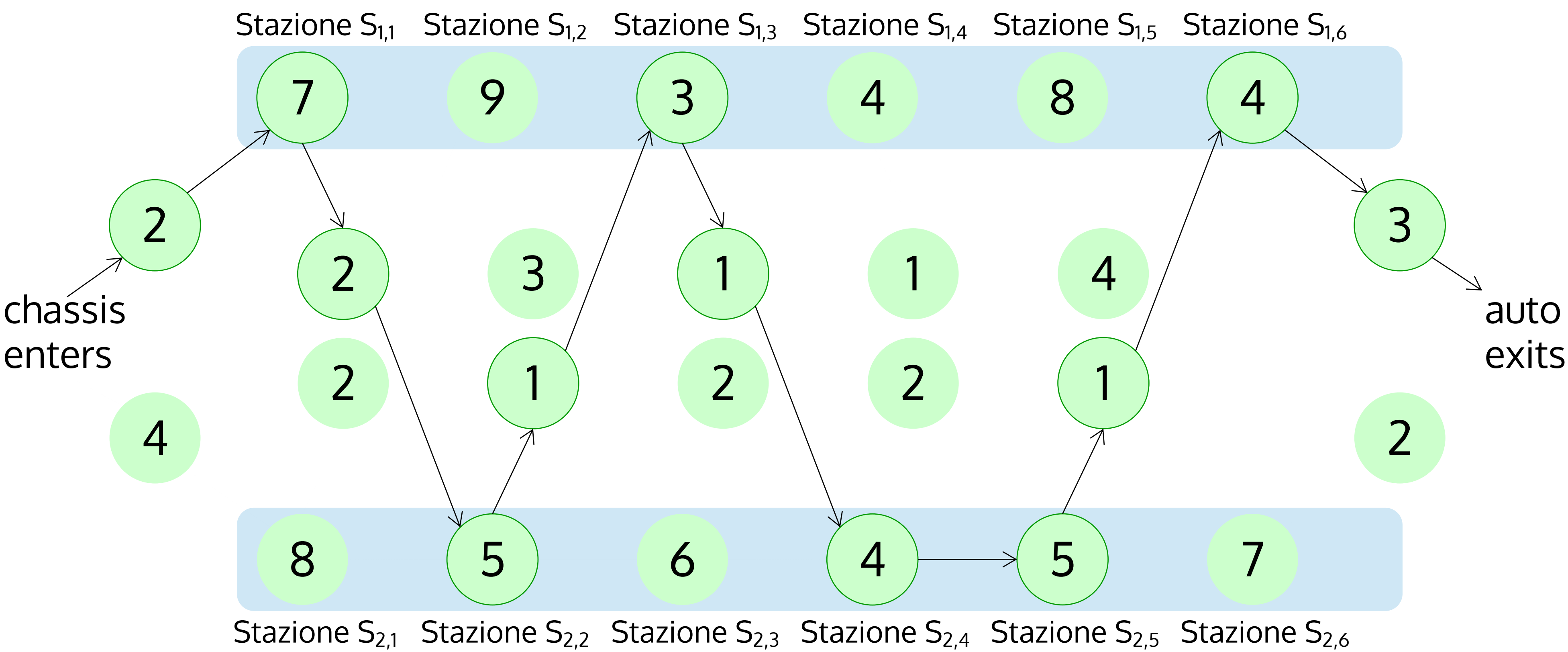
$l^*=1$

- Usando l^* e $l_i[j]$ si riesce a tracciare il percorso più veloce

Calcolo del valore ottimo

- Sfruttando le equazioni ricorsive ricavate, potremmo scrivere un algoritmo ricorsivo
- Avrebbe un tempo di esecuzione esponenziale!
  - Sia $r_i(j)$ il numero di chiamate ricorsive per calcolare $f_i[j]$
  - $r_1(n) = r_2(n) = 1$
  - $r_1(j) = r_2(j) = r_1(j+1) + r_2(j+1)$ per $j=1,2,\ldots,n-1$
  - Con il metodo di sostituzione si verifica che $r_i(j) = 2^{n-j}$
  - $f_1[1]$ viene chiamata 2^{n-1} volte!
- Si vede che il numero totale di chiamate ricorsive è $\Theta(2^n)$
- Calcolando invece $f_1[j]$ e $f_2[j]$ a partire da $j=1$ riusciamo ad ottenere un tempo di esecuzione $\Theta(n)$

Catene di montaggio



j	1	2	3	4	5	6
$f_1[j]$						
$f_2[j]$						

$f^*=38$

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

$l^*=1$

- Usando l^* e $l_i[j]$ si riesce a tracciare il percorso più veloce

Pseudo-codice

```
Fastest-Way (a,t,e,x,n)
f1[1] ← e1+a1,1
f2[1] ← e2+a2,1
for j ← 2 to n
do if f1[j-1] <= f2[j-1]+t2,j-1
    then f1[j] ← f1[j-1]+a1,j
        l1[j] ← 1
    else f1[j] ← f2[j-1]+t2,j-1+a1,j
        l1[j] ← 2
if f2[j-1] <= f1[j-1]+t1,j-1
    then f2[j] ← f2[j-1]+a2,j
        l2[j] ← 2
    else f2[j] ← f1[j-1]+t1,j-1+a2,j
        l2[j] ← 1
if f1[n]+x1 <= f2[n]+x2
then f* = f1[n]+x1
    l* ← 1
else f* = f2[n]+x2
    l* ← 2
```

Il tempo di esecuzione è $\Theta(n)$

Pseudo-codice

Procedura per stampare la lista di stazioni attraversate dal percorso più breve (in ordine inverso)

```
Print-Stations (l,n)
i ← l*
print “line ” i “, station ” n
for j ← n downto 2
do i ← li[j]
print “line ” i “, station ” j-1
```

Programmazione dinamica

- È conveniente utilizzare la programmazione dinamica quando
  - Il problema esibisce una “sottostruttura” ottima
  - Si presentano sottoproblemi “sovrapposti”

Sottostruttura ottima

- Il problema esibisce una sottostruttura ottima quando una soluzione ottima del problema contiene al suo interno soluzioni ottime di sottoproblemi
- Questo è un segno che l'approccio della programmazione dinamica *può* essere impiegato
  - Ma si potrebbero impiegare anche divide-et-impera e approccio greedy
- La sottostruttura ottima varia da problema a problema in termini
  - del numero di sottoproblemi presenti nella soluzione ottima del problema originario (1 nel nostro esempio)
  - del numero di scelte che si hanno nel determinare quale/i sottoproblema/i è presente nella soluzione ottima (2 nel nostro esempio)

Sottostruttura ottima

- Il tempo di esecuzione di un algoritmo di programmazione dinamica è tipicamente il prodotto di due fattori:
  - Numero complessivo di sottoproblemi
  - Numero di sottoproblemi tra cui scegliere per ottenere la soluzione al problema originario
- La programmazione dinamica utilizza un approccio bottom-up
  - Gli algoritmi greedy un approccio top-down

Sottoproblemi sovrapposti

- È conveniente utilizzare la programmazione dinamica quando un algoritmo ricorsivo invocherebbe più volte uno stesso sottoproblema
  - Se ciò non accade, l'approccio divide-et-impera è adeguato
- La programmazione dinamica risolve ogni sottoproblema una sola volta e utilizza il valore calcolato quando si ripresenta lo stesso sottoproblema nell'approccio bottom-up