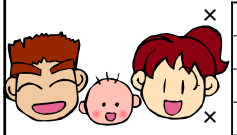
データ構造とアルゴリズムII

第2回
動的計画法1

48

パレート効率性

- パレート支配：状態xとx'を比較して，すべてのプレイヤがxの方がx'より望ましい (あるいは同じ) と思っており，少なくとも一人がxの方が厳密に良いと思っている場合，xはx'をパレート支配する
  - x'の代わりにxを選ぶことに反対するプレイヤはいない
- パレート効率的：他のどんな状態にもパレート支配されない状態はパレート効率的
  - いずれかの参加者の効用を犠牲にすることなしには，他の参加者の効用を向上することができない状態



映画	2	2	2
デパート	2	2	5
公園	2	3	1
家にいる	1	1	1

49

パレート効率性 (続き)

- 1000円を二人で分ける
- 捨てても構わない
- (0, 1000), (x, 1000-x), (1000, 0)のいずれもパレート効率的
- 500円捨てて (250, 250)よりも，(250+x, 750-x)の方が良いことは二人とも合意可能

50

パレート効率性 (続き)

- そもそも異なるプレイヤの効用が比較できるか，同じ尺度で計れるかに関しては議論がある
- パレート効率性の定義は，プレイヤ間の効用が比較できない場合でも適用可能
- 社会的な望ましい状態に関する(最低限の?) 要求条件
  - x'がパレート効率的でなければ，別の状態xがあり，全員がxの方が良い (少なくとも同じ) と思っている

51

パレート効率性

- DAの結果は女性にとってパレート効率的?
- 不安的なペアの存在／男性の選好を無視すれば，女性全員がより幸せになる方法が存在

	John	Ken	Lee
first	Carol	Carol	Becky
second	Alice	Alice	Carol
third	Becky	Becky	Alice

	Alice	Becky	Carol
first	John	John	Lee
second	Ken	Lee	John
third	Lee	Ken	Ken

52

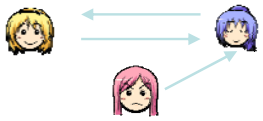
Top Trading Cycles メカニズム (Shapley and Scarf 1974)

- まず，男性を各女性にランダムに割り当てる (初期保有)．
- 各女性は，最も好みの男性を保有する女性を指す (自分自身を指してもよい)．
- 少なくとも一つの指指しのサイクルが存在するので，そのサイクルに従って，サイクルに含まれる女性に，好みの男性を割り当てる．これらの女性はマーケットから退出する．
- 残りの女性間で上記の手続きを，すべての女性がマーケットから退出するまで繰り返す．

53

例 (TTC)

	Alice	Becky	Carol
first	John	John	Lee
second	Ken	Lee	John
third	Lee	Ken	Ken



54

TTCの性質

- 誘因両立性:
  - 自分に至る指差しのパスが存在すれば，そのパスに含まれる女性が保有する男性を得ることが可能.
  - そのようなパスは，自分がマーケットから退出するまで存在し続ける＝後でも入手可能
  - よって，今の第一希望を諦めて，これらの男性を手に入ようとする必要はない.
- パレート効率性:
  - 最初にできたサイクルでは，すべての女性は第一希望の男性を得ている.
  - 二回目のサイクルで，第二希望を得た女性を第一希望にするには，最初のサイクルの女性の誰かを犠牲にしないといけない.

55

TTCの両方向マッチングへの適用方法

メカニズム:

- 各学生は，最も好む学校を指差す.
- 各学校は，最も好む学生を指差す.
- 少なくとも一つサイクルが存在.
- サイクルに含まれる学生は，指差している学校の席を得てマーケットから退出する. 各学校は，残りの席がなくなればマーケットから退出する.
- 残りの学生／学校間で上記の手続きを繰り返し適用する.

56

TTCの性質: Two-sided

- 学生にとって誘因両立的.
- 学生にとってパレート効率的. パレート効率的なら自動的に非浪費性は満足
- 結果は不安定. 一般に安定性とパレート効率性は両立不可能

57

15. 動的計画法

58

動的計画法

(dynamic programming)

||

部分問題をボトムアップに解いて統合する

59

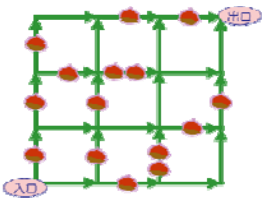
動的計画法のアルゴリズム

1. 最適解の構造を特徴づける（部分問題最適性）
2. 最適解の値を再帰的に定義する
3. ボトムアップに最適解の値を求める
4. 最適解を構成する

60

簡単な例：栗拾い

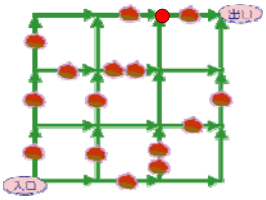
- 左の図のような格子状に道がある栗林で，拾える栗の個数を最大化する経路を求めたい
- ある交差点からは，右か上にしか進めない
- n×nの栗林だと，経路の総数は$\frac{(2n)!}{n!n!}$なので，総当たりだとnが大きくなると絶望的



61

簡単な例：栗拾い

- 部分問題最適性（最適性の原理）：もし最適解が，ある交差点を通るなら，その最適解の，入口からその交差点までの経路（部分問題の解）は，入口からその交差点までに拾える栗の数を最大化している
- そうでなければ，入口からその交差点までの経路を入れ替えれば，より良い解が得られて矛盾

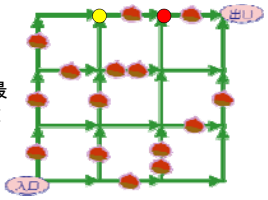


62

簡単な例：栗拾い

- ある交差点までに拾える栗の数の最大値を求めることを（副）目標とする
- 交差点 i までの最適解が，i に至る一歩手前で，交差点 j を通ることが分かっているなら，j までの最適解に，$j-i$ 間で拾える栗の個数を足せばよい
- しかし，そのような交差点がどれかは，あらかじめ分かっていない

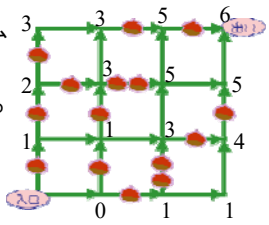
⇒ 全部試して，最も良いものを選択すればよい（最適解の再帰的な定義）



63

簡単な例：栗拾い

- 入口に近い交差点から順に，その交差点までに拾える栗の数の最大値を計算していけばよい（ボトムアップに最適解を求める）
- $(n+1)^2$個の交差点で上記の計算をするだけで最大値が得られる
- 上記の交差点の最大値から最適経路を構成するのは簡単



64

演習：最長増加部分列

- 並んでいる木から，いくつかを伐採し，残った木の高さが，左から右に順に高くなるようにしたい
- なるべく多くの木を残すにはどうしたらよい?
- 全ての部分列の数は2^n

65

最長増加部分列：ヒント

- ある木 t が最長増加部分列に含まれていると仮定
- 最長増加部分列において，t の左側は，元の木の列で，tより短かく，tより左側にある木に関する最長増加部分列になっているはず（部分問題最適性）
- その部分列に t を加えたら，t を含む最長増加部分列が得られる（最適解の再帰的な構造）

66

最長増加部分列：解答

木の高さ:0	4	1	6	2	8	5	7	3
列の長さ:0	1	1	2	2	3	3	4	3

67

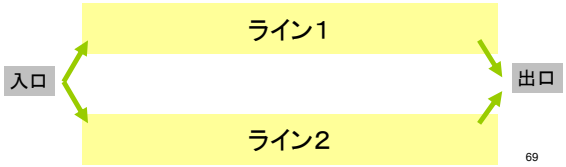
例題

組み立てライン
スケジューリング

68

問題

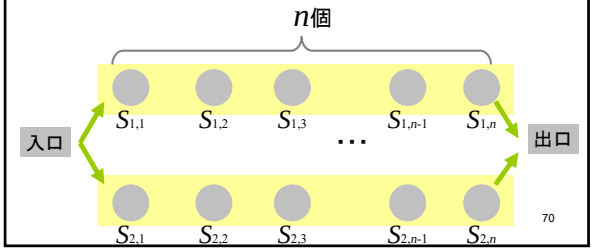
- 車を組み立てる2つのラインがある



69

問題

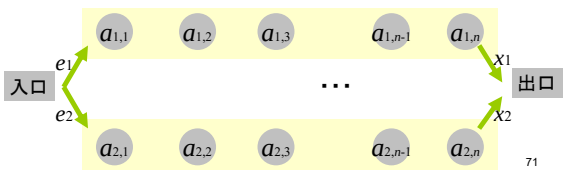
- 車を組み立てる2つのラインがある
- 各ラインにはn個のステーション $S_{1,j}$， $S_{2,j}$ がある



70

問題

- 車を組み立てる2つのラインがある
- 各ラインにはn個のステーション $S_{1,j}$， $S_{2,j}$ がある
- ステーション$S_{i,j}$では組立に$a_{i,j}$時間かかる

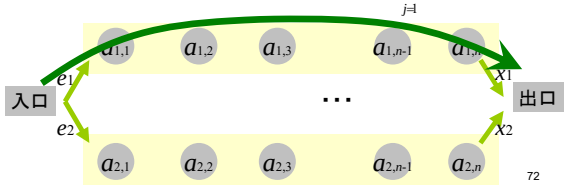


71

問題

- 車を組み立てる2つのラインがある
- 各ラインにはn個のステーション $S_{1,j}$, $S_{2,j}$ がある
- ステーション$S_{i,j}$では組立に$a_{i,j}$時間かかる

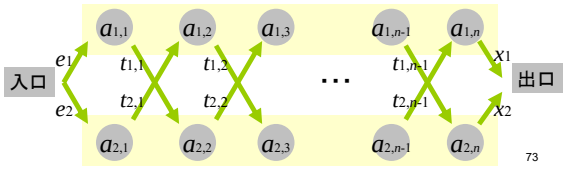
(例)ライン1を通過する時間 $= e_1 + \sum_{j=1}^{n-1} a_{1,j} + x_1$



72

問題

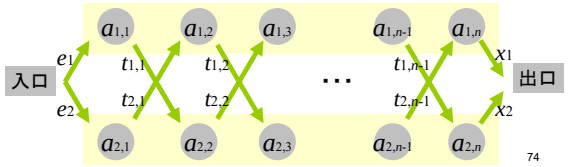
- 車を組み立てる2つのラインがある
- 各ラインにはn個のステーション $S_{1,j}$, $S_{2,j}$ がある
- ステーション$S_{i,j}$では組立に$a_{i,j}$時間かかる
- ラインを変えるには$t_{i,j}$時間かかる



73

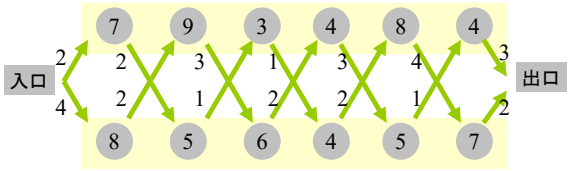
問題

- 車を組み立てる2つのラインがある
- 各ラインにはn個のステーション $S_{1,j}$, $S_{2,j}$ がある
- ステーション$S_{i,j}$では組立に$a_{i,j}$時間かかる
- ラインを変えるには$t_{i,j}$時間かかる
  → **最短時間の順路を求める**



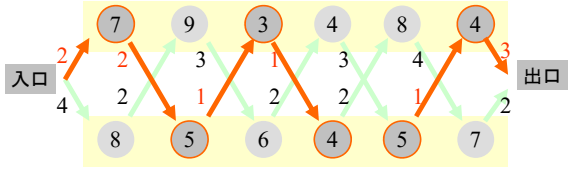
74

例



75

例



2 + 7 + 2 + 5 + 1 + 3 + 1 + 4 + 5 + 1 + 4 + 3
=38

76

総当り作戦の破綻

- ある道筋のコストを計算するには
  →$O(n)$
- 全ての道筋は
  →2^n 通り

総当りには$\Omega(2^n)$必要

※$\Omega(2^n)$ = 少なくとも 2^n の計算量を要する

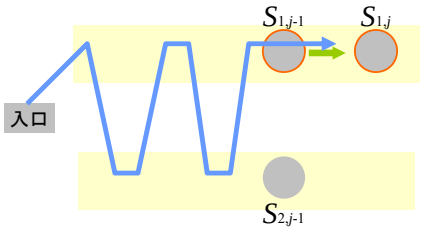
77

アルゴリズムのステップ1:
最適解の構造を
特徴づける

78

ステーション$S_{1,j}$を最速で終了する順路が

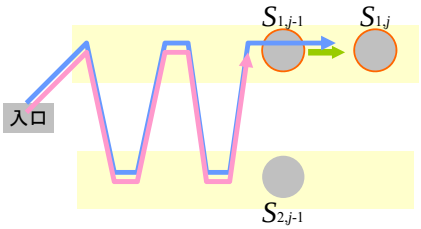
- $S_{1,j-1}$を経由していた場合



79

ステーション$S_{1,j}$を最速で終了する順路が

- $S_{1,j-1}$を経由していた場合
  - $S_{1,j-1}$を最速で終了する順路を辿ってきたはず

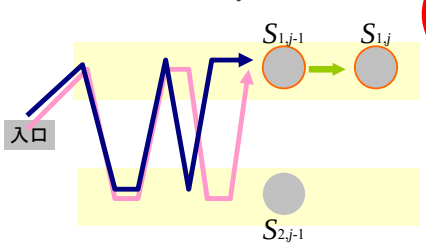


80

なぜならば

- もっと速く$S_{1,j-1}$に到達する道があるなら,
  この順路を使って $S_{1,j}$ にもっと速く着ける

矛盾



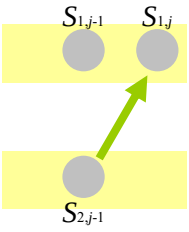
81

同様に…

82

ステーション$S_{1,j}$を最速で終了する順路が

- $S_{1,j-1}$を経由していた場合
  - $S_{1,j-1}$を最速で終了する順路を辿ってきた
- $S_{2,j-1}$を経由していた場合
  - $S_{2,j-1}$を最速で終了する順路を辿ってきた
- j=1のとき
  - 1通り

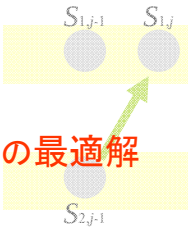


83

ステーション$S_{1,j}$を最速で終了する順路が
問題の最適解

- $S_{1,j-1}$を経由していた場合
  - $S_{1,j-1}$を最速で終了する順路を辿ってきた
- $S_{2,j-1}$を経由していた場合
  - $S_{2,j-1}$を最速で終了する順路を辿ってきた
- j=1のとき
  - 1通り

部分問題の最適解



84

問題の最適解に
部分問題の最適解が含まれる
||

部分問題最適性

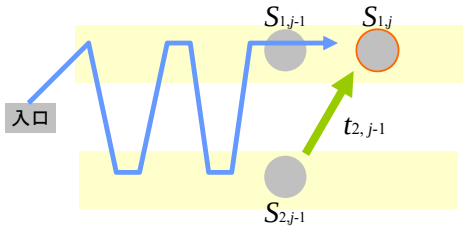
85

アルゴリズムのステップ2.
最適解の値を
再帰的に定義する

86

最適解の値

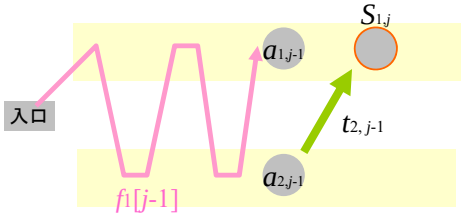
$f_i[j]$ = 入口から$S_{i,j}$までの最短時間



87

最適解の値

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



88

最適解の値

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

同様に…

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

求めたい最適解の値は…

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

89

アルゴリズムのステップ3:
（ボトムアップに）
最適解の値を求める

90

再帰をそのまま解くと･･･$O(2^n)$

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

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

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

91

再帰をそのまま解くと･･･$O(2^n)$

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

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

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

→ ボトムアップ(昇順)に解く

92

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

```
f1[1] ← e1
f2[1] ← e2
for j ← 2 to n
    do if f1[j−1] + a1,j ≤ f2[j−1] + t2,j−1 + a1,j
       then f1[j] ← f1[j−1] + a1,j
            l1[j] ← 1
       else f1[j] ← f2[j−1] + t2,j−1 + a1,j
            l1[j] ← 1
       if f2[j−1] + a2,j ≤ f1[j−1] + t1,j−1 + a2,j
       then f2[j] ← f2[j−1] + a2,j
            l2[j] ← 1
       else f2[j] ← f1[j−1] + t1,j−1 + a2,j
            l2[j] ← 1
if f1[n] + a1,n + x1 ≤ f2[n] + a2,n + x2
    then f* ← f1[n] + a1,n + x1
         l* ← 1
    else f* ← f2[n] + a2,n + x2
         l* ← 2
```

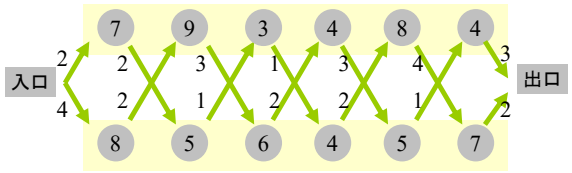
初期値を設定

$f_i[j]$を計算

最適解の値を計算

93

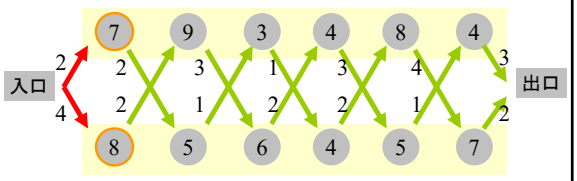
例



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

94

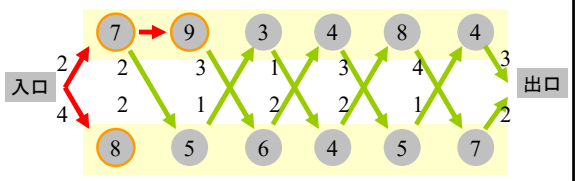
例



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

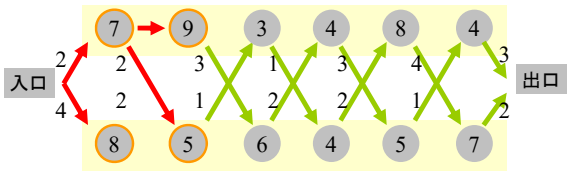
95

例



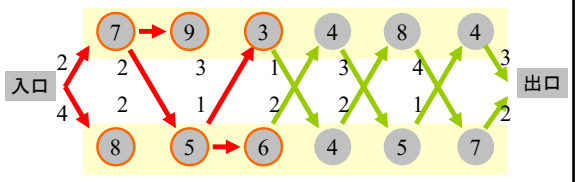
j	1	2	3	4	5	6
$f_1[j]$	2	9				
$f_2[j]$	4					

例



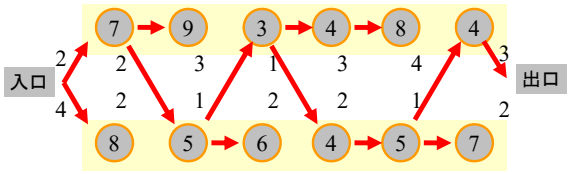
j	1	2	3	4	5	6
$f_1[j]$	2	9				
$f_2[j]$	4	11				

例



j	1	2	3	4	5	6
$f_1[j]$	2	9	17			
$f_2[j]$	4	11	16			

例



j	1	2	3	4	5	6	
$f_1[j]$	2	9	17	20	24	31	38
$f_2[j]$	4	11	16	21	25	30	39

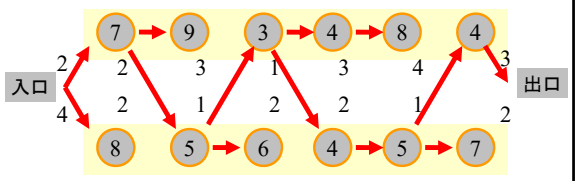
アルゴリズムのステップ4：最適解を構成する

- l^*, $l[j]$ から順路を出力する

PRINT-STATIONS(l, l^*, n)

$i \leftarrow l^*$
print "ライン"+i+"ステーション"+n
for $j \leftarrow n$ **downto** 2
  do $i \leftarrow l_i[j]$
    print "ライン"+i+"ステーション"+(j−1)

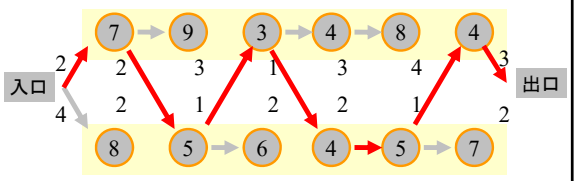
例



j	1	2	3	4	5	6	
$f_1[j]$	2	9	17	20	24	31	38
$f_2[j]$	4	11	16	21	25	30	39

102

例



j	1	2	3	4	5	6	
$f_1[j]$	2	9	17	20	24	31	38
$f_2[j]$	4	11	16	21	25	30	39

103

動的計画法のまとめ

1. 最適解の構造を特徴づける
   - 部分問題最適性がポイント
2. 最適解の値を再帰的に定義する
3. ボトムアップに最適解の値を求める
4. 最適解を構成する
   - 3の手続き中で実行可能な場合も多い

104