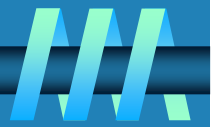
Lecture 9

Dyna c ro ra n

Dynamic Programming

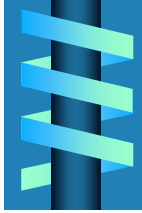


D*ynamic Programming* s a n ra a or t d s n t c n qu or so v n prob s d n d by r curr nc s w t ov r app n subprob s

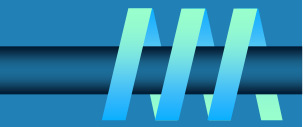
- Inv nt d by A r can at at c an c ard B an n t s to so v opt zat on prob s and at r ass at d by C

- ¸ ro ra n r ans ¸p ann n

- Ma n d a
  - s t up a r curr nc r at n a so ut on to a ar r nstanc to so ut ons o so s a r nstanc s so v s a r nstanc s onc
  - r cord so ut ons n a tab
  - xtract so ut on to t n t a nstanc ro t at tab



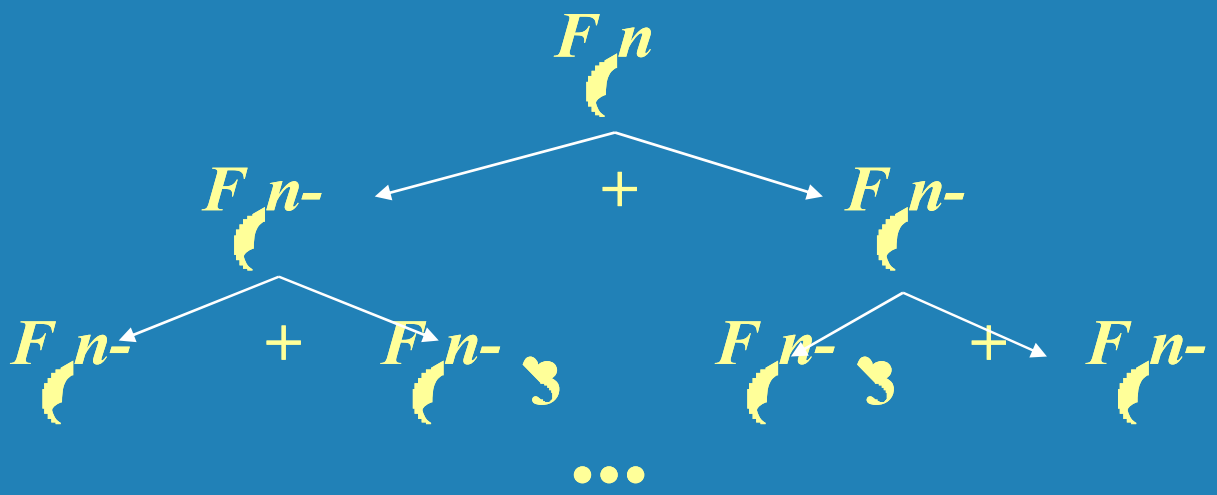
Example: Fibonacci numbers



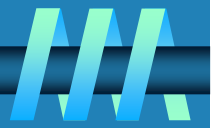
- ca d n t on o F bonacc nu b rs

$$F_n = F_{n-1} + F_{n-2}$$

- Co put n t n^{th} F bonacc nu b r r curs v y top down



Example: Fibonacci numbers (cont.)



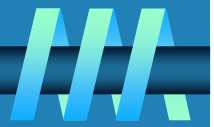
Computing the n^{th} Fibonacci number using bottom-up iteration and recording results:

$F(0) = 0$
$F(1) = 1$
$F(2) = 1+0 = 1$
…
$F(n-2) =$
$F(n-1) =$
$F(n) = F(n-1) + F(n-2)$

0	1	1	. . .	$F(n-2)$	$F(n-1)$	$F(n)$

Efficiency:
- time
- space

Examples of DP algorithms



- Co put n a b no a co c nt
- ars a s a or t or trans t v c osur
- F oyd s a or t or a pa rs s ort st pat s
- Construct n an opt a b nary s arc tr
- o nstanc s o d cu t d scr t opt zat on prob s
  trav n sa s an
  napsac

Computing a binomial coefficient by DP

Binomial coefficients are coefficients of the binomial formula:

$$(a + b)^n = C(n,0)a^n b^0 + \ldots + C(n,k)a^{n-k}b^k + \ldots + C(n,n)a^0 b^n$$

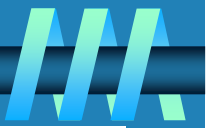
Recurrence: $C(n,k) = C(n\text{-}1,k) + C(n\text{-}1,k\text{-}1)$ for $n > k > 0$

$C(n,0) = 1, \quad C(n,n) = 1$ for $n \geq 0$

Value of $C(n,k)$ can be computed by filling a table:

	0	1	2	. . .	k-1	k
0	1					
1	1	1				
.						
.						
.						
n-1					$C(n\text{-}1,k\text{-}1)$	$C(n\text{-}1,k)$
n						$C(n,k)$

Computing $C(n,k)$: pseudocode and analysis



ALGORITHM *Binomial*(n, k)

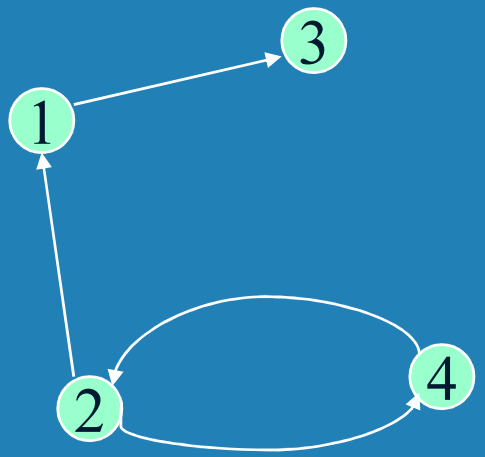
//Computes $C(n, k)$ by the dynamic programming algorithm
//Input: A pair of nonnegative integers $n \geq k \geq 0$
//Output: The value of $C(n, k)$
for $i \leftarrow 0$ **to** n **do**
    for $j \leftarrow 0$ **to** $\min(i, k)$ **do**
        if $j = 0$ **or** $j = i$
            $C[i, j] \leftarrow 1$
        else $C[i, j] \leftarrow C[i-1, j-1] + C[i-1, j]$
return $C[n, k]$

c rrcy Θ *nk*

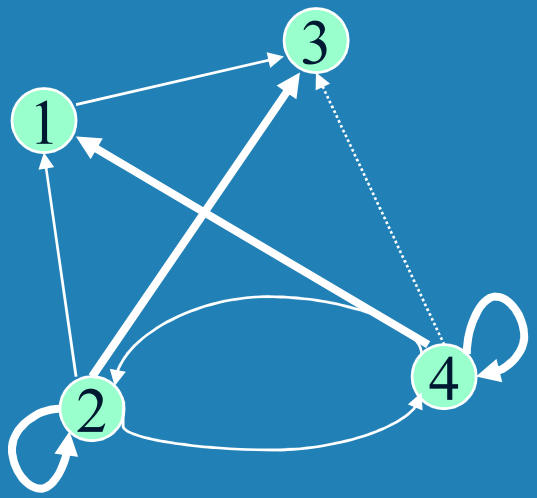
pac c rrcy Θ *nk*

Warshall's Algorithm: Transitive Closure

- Co put s t trans t v c osur o a r at on
- A t rnat v· y x st nc o a nontr v a pat s n a d rap
- Exa p o trans t v c osur



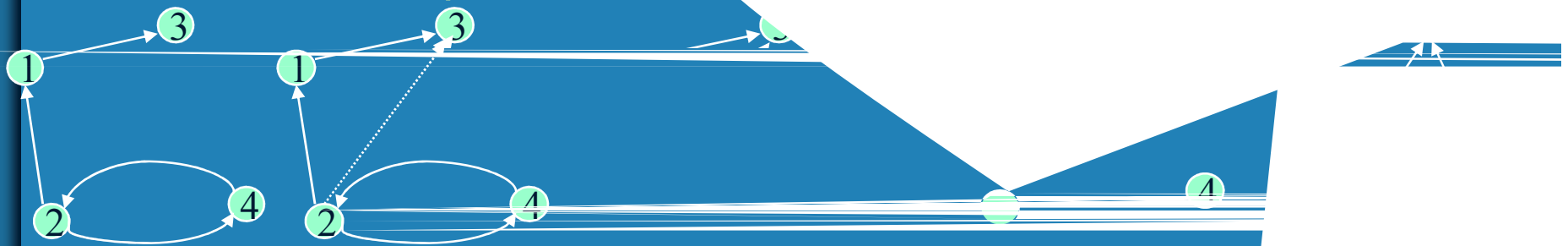
0 0 1 0
1 0 0 1
0 0 0 0
0 1 0 0



0 0 1 0
1 **1** **1** 1
0 0 0 0
1 1 **1** **1**

Warshall's Algorithm

Constructs trans t v c osur T as t ast
o n by n atr c s $R^{(0)}$. $R^{(k)}$
$R^{(k)}$ i j t r s non
v rt c s a ow d as nt r
Not t at $R^{(0)}$ A ad ac nc



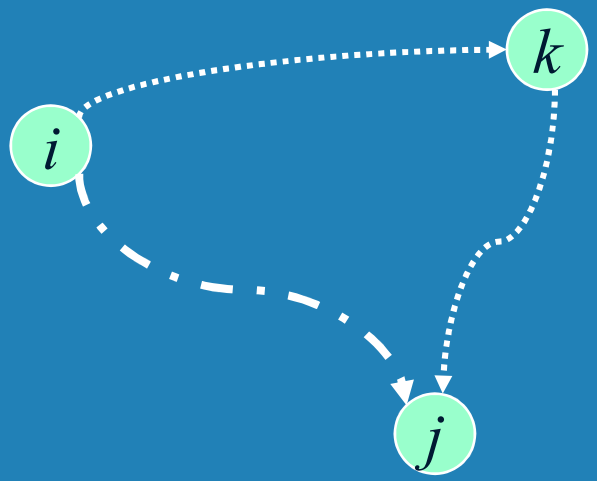
Warshall's Algorithm (recurrence)

- n t *k*-t t rat on t a or t d t r n s or v ry pa r o v rt c s *i, j* a pat x sts ro *i* and *j* w t ust v rt c s . *k* a ow d as nt r d at

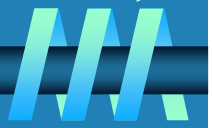
$$R^{(k)}\ i,j \begin{cases} R^{(k-1)}\ i,j \\ \text{or} \\ R^{(k-1)}\ i,k \ \text{and}\ R^{(k-1)}\ k,j \end{cases}$$

pat us n ust . *k*-

pat ro *i* to *k*
and ro *k* to *i*
us n ust . *k*-



Warshall's Algorithm (matrix generation)



curr nc r at n nts $R^{(k)}$ to nts o $R^{(k-1)}$ s

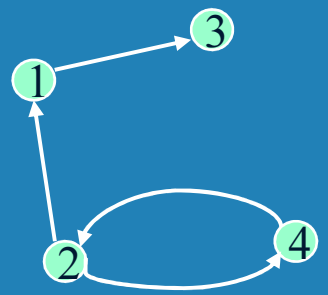
$R^{(k)}$ i,j $R^{(k-1)}$ i,j or $R^{(k-1)}$ i,k and $R^{(k-1)}$ k,j

It p s t o ow n ru s or n rat n $R^{(k)}$ ro $R^{(k-1)}$

u I an nt n row i and co u n j s n $R^{(k-1)}$
t r a ns n $R^{(k)}$

u I an nt n row i and co u n j s n $R^{(k-1)}$
t as to b c an d to n $R^{(k)}$ and on y
t nt n ts row i and co u n k and t nt
n ts co u n j and row k ar bot s n $R^{(k-1)}$

Warshall's Algorithm (example)



$$R^{(0)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad R^{(1)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$R^{(2)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ & 1 & & \end{bmatrix} \qquad R^{(3)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \qquad R^{(4)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Warshall's Algorithm (pseudocode and analysis)

ALGORITHM *Warshall*$(A[1..n, 1..n])$

```
//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph
R^(0) ← A
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            R^(k)[i, j] ← R^(k−1)[i, j] or (R^(k−1)[i, k] and R^(k−1)[k, j])
return R^(n)
```

c rcy $\Theta(n^3)$

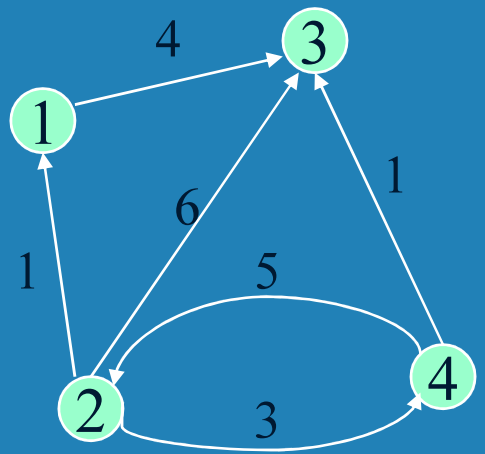
pac c rcy Matr c s can b wr tt n ov r t r pr d c ssors

Floyd's Algorithm: All pairs shortest paths

rob · In a w t d d rap nd s ort st pat s b tw n v ry pa r o v rt c s

a d a construct so ut on t rou s r s o atr c s $D^{(0)}$. $D^{(n)}$ us n ncr as n subs ts o t v rt c s a ow d as nt r d at

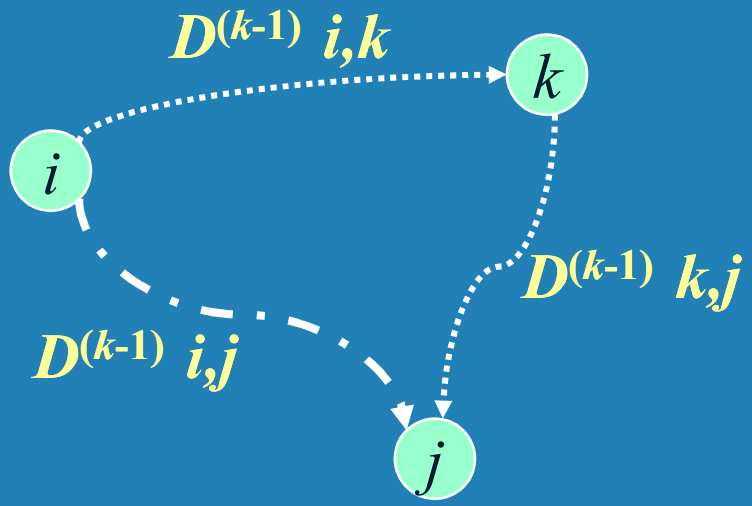
Exa p



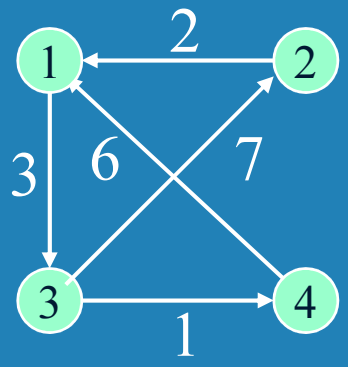
Floyd's Algorithm (matrix generation)

- n t *k*-t t rat on t a or t d t r n s s ort st pat s b tw n v ry pa r o v rt c s *i, j* t at us on y v rt c s a on . *k* as nt r d at

$D^{(k)}\ i,j$ n $D^{(k-1)}\ i,j$ $D^{(k-1)}\ i,k$ $D^{(k-1)}\ k,j$



Floyd's Algorithm (example)



$$D^{(0)} = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \qquad D^{(1)} = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \qquad D^{(3)} = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 9 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix}$$

Floyd's Algorithm (pseudocode and analysis)

ALGORITHM *Floyd*$(W[1..n, 1..n])$

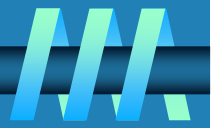
```
//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
D ← W //is not necessary if W can be overwritten
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
return D
```

c rcy $\Theta(n^3)$

pac c rcy Matr c s can b wr tt n ov r t r pr d c ssors

Not ort st pat s t s v s can b ound too

Optimal Binary Search Trees



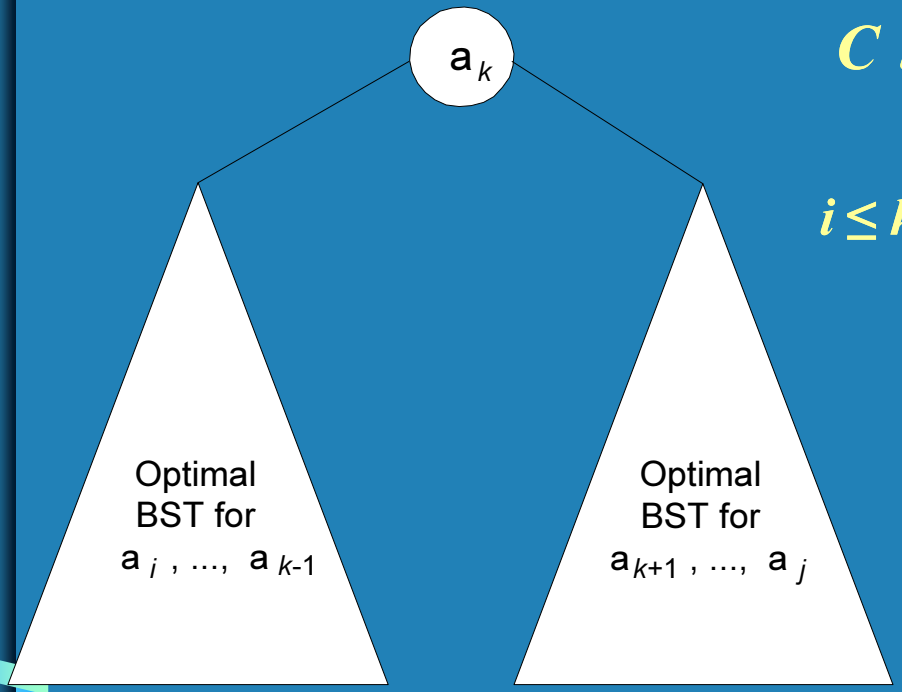
rob · G v n n ys a_1 . a_n and probab t s p_1 . p_n
s arc n or t nd a B w t a n u
av ra nu b r o co par sons n succ ss u s arc .

nc tota nu b r o B s w t n nod s s v n by
C n n / n w c rows xpon nt a y brut orc s op ss.

Exa p at s an opt a B or ys A B C and D w t
s arc probab t s . . . and . r sp ct v y .

DP for Optimal BST Problem

L t $C(i,j)$ b n u av ra nu b r o co par sons ad n
i,j opt a B or ys a_i . a_j w r i j n.
Cons d r opt a B a on a B s w t so a_k i k j
as t r root i,j s t b st a on t ,



$C(i,j)$

n p_k .
$i \le k \le j$

k-1
$\sum p_s$ v a_s n i,k-
$s = i$

$^{j} p_s$ v a_s n $k+$,j
s =k+1

Example: key	*A*	*B*	*C*	*D*
probability	0.1	0.2	0.4	0.3

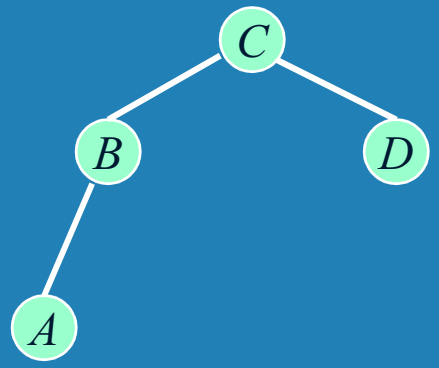
tab s b ow ar d d a ona by d a ona t t on s d
us n t r curr nc

$$C[i,j] = \min_{i \le k \le j} \{ C[i,k] + C[k,j] \} + \sum_{s=i}^{j} p_s, \quad C[i,i] = p_i$$

t r t on or tr s, roots r cords *k*s va u s v n t n a

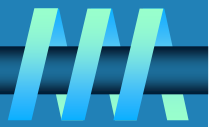
i \ *j*	0	1	2	3	4
1	0	.1	.4	1.1	1.7
2		0	.2	.8	1.4
3			0	.4	1.0
4				0	.3
5					0

i \ *j*	0	1	2	3	4
1		1	2	3	3
2			2	3	3
3				3	3
4					4
5					



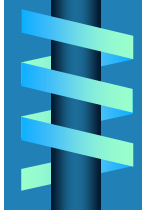
opt a B

Optimal Binary Search Trees



ALGORITHM *OptimalBST*$(P[1..n])$

```
//Finds an optimal binary search tree by dynamic programming
//Input: An array P[1..n] of search probabilities for a sorted list of n keys
//Output: Average number of comparisons in successful searches in the
//        optimal BST and table R of subtrees' roots in the optimal BST
for i ← 1 to n do
    C[i, i − 1] ← 0
    C[i, i] ← P[i]
    R[i, i] ← i
C[n + 1, n] ← 0
for d ← 1 to n − 1 do //diagonal count
    for i ← 1 to n − d do
        j ← i + d
        minval ← ∞
        for k ← i to j do
            if C[i, k − 1] + C[k + 1, j] < minval
                minval ← C[i, k − 1] + C[k + 1, j]; kmin ← k
        R[i, j] ← kmin
        sum ← P[i]; for s ← i + 1 to j do sum ← sum + P[s]
        C[i, j] ← minval + sum
return C[1, n], R
```



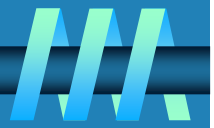
Analysis DP for Optimal BST Problem

c rrcy $\Theta(n^3)$ but can b r duc d to $\Theta(n^2)$ by ta n advanta o onoton c ty o ntr s n t root tab $R_{i,j}$ s a ways n t ran b tw n $R_{i,j}$ and $R_{i,j}$

pac c rrcy $\Theta(n^2)$

M t od can b xp nd d to nc ud unsucc ss u s arc s

Knapsack Problem by DP



G v n n t s o

nt r w ·ts w_1 w_2 … w_n

va u s v_1 v_2 … v_n

a napsac o nt r capac ty W

nd ost va uab subs t o t t s t at t nto t napsac

Cons d r nstanc d n d by rst i t s and capac ty j $j \leq W$.

L t $V\,i\,j$ b opt a va u o suc nstanc , n

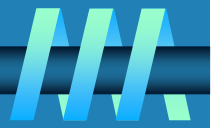
ax $V\,i\,j$ v_i $V\,i\,j$- w_i j- $w_i \geq 0$

$V\,i\,j$

$V\,i\,j$ j- $w_i < 0$

In t a cond t σns V j and $V\,i$

Knapsack Problem by DP (example)



Exa p Knapsac o capac ty W

t w t va u

capac ty j

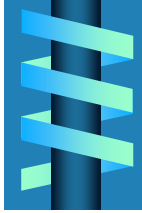
w_1 v_1=

w_2 v_2=

w_3 v_3=

w_4 v_4=

?



Excersie

ct on ,

ct on ,

ct on , 3 7

ct on ,