

Vistula, IT Faculty, 2014

# Algorithms and Complexity

Dmitry A. Zaitsev

<http://daze.ho.ua>

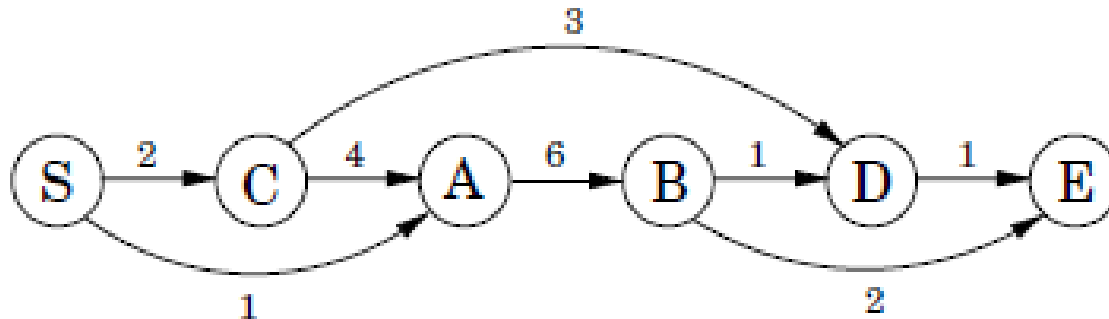
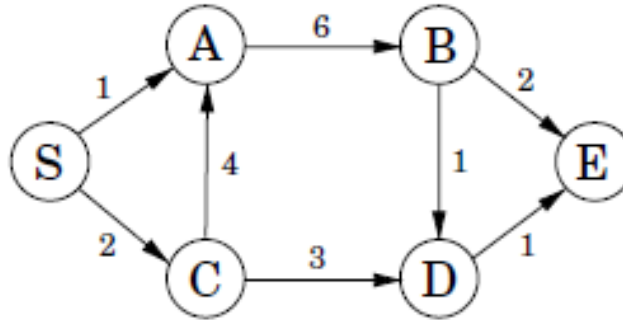
## Lecture 14:

## Dynamic programming

## Principles of DP

- Collection of subproblems
- Process them one by one
- Store intermediate results in a table
- From small to large problems
- Use (and reuse) previously tabulated results
- Recover a solution from table

## Paths in directed acyclic graphs



$$\text{dist}(v) = \min_{(u,v) \in E} \{ \text{dist}(u) + w(u,v) \}$$

! reuse dist(C) for A and D etc

# Fibonacci numbers

```
long ft[1000];  
int ft_last;
```

```
long fib( int n )  
{  
    int i;  
    if( n > ft_last )  
    {  
        for(i=ft_last+1;i<=n;i++)  
            ft[i]=ft[i-2]+ft[i-1];  
        ft_last = n;  
    }  
    return(ft[n]);  
}
```

```
main()  
{  
    int n;  
    ft[0]=1L;  
    ft[1]=1L;  
    ft_last = 1;  
    while(n!=0)  
    {  
        scanf("%d",&n);  
        if(n==0)break;  
        printf("f(%d)=%d\n",n,fib(n));  
    }  
}
```

# Path in a triangle of numbers

i \ j	0	1	2	3	4
0	3				
1	6	7			
2	3	4	5		
3	6	9	3	1	
4	2	9	1	4	3

$O(n^2)$   
instead  
 $O(2^n)$

$B[i][j]$ :

Matrix of the best sums –  $b[n][n]$

i \ j	0	1	2	3	4
0	3				
1	9	10			
2	12	14	15		
3	18	23	18	16	
4	20	32	24	22	19

If( $j==0$ )  
 $b[i][j]=a[i][j]+b[i-1][j];$

If( $j==i$ )  
 $b[i][j]=a[i][j]+b[i-1][j-1]$

else  
max(  
( $a[i][j]+b[i-1][j-1]$ ),  
( $a[i][j]+b[i-1][j]$ )))

# Matrix multiplication

$$M = M_1 \cdot M_2 \cdot \dots \cdot M_n$$

$A_i$  matrix of  $r_{i-1}$  rows and  $r_i$  columns:  $r_{i-1} \times r_i$

$$A = B \cdot C, a_{i,j} = \sum_{k=1}^q b_{i,k} \cdot c_{k,j}$$

To multiply  $p \times q$  by  $q \times r$   
 $p \cdot q \cdot r$  operations are required

$$M = M_1 \cdot M_2 \cdot M_3 \cdot M_4 =$$
$$M_1 \cdot (M_2 \cdot (M_3 \cdot M_4)) = (M_1 \cdot (M_2 \cdot M_3)) \cdot M_4$$

$r_i$ : 10,20,50,1,100

125000 versus 2200 operations

## Matrix multiplication – DP solution

Let  $m_{i,j}$  - the minimal cost of calculating  $M_i \cdot M_{i+1} \cdot \dots \cdot M_j$ ,  $1 \leq i \leq j \leq n$

$$m_{i,j} = \begin{cases} 0, i = j \\ \min_{i \leq k < j} (m_{i,k} + m_{k,j} + r_{i-1} \cdot r_k \cdot r_j), i < j \end{cases}$$

$$M_i \cdot M_{i+1} \cdot \dots \cdot M_j = (M_i \cdot M_{i+1} \cdot \dots \cdot M_k) \cdot (M_{k+1} \cdot M_{k+2} \cdot \dots \cdot M_j)$$

$m_{i,k}$  minimal cost of  $M' = M_i \cdot M_{i+1} \cdot \dots \cdot M_k$

$m_{i,k}$  minimal cost of  $M'' = M_i \cdot M_{i+1} \cdot \dots \cdot M_j$

$r_{i-1} \cdot r_k \cdot r_j$  cost of  $M' \cdot M''$

## Matrix multiplication – DP algorithm

```
for(i=1;i<=n;i++) m[i][i]=0;
for(l=1;l<=n-1;l++)
{
    for(i=1;i<=n-l;i++)
    {
        j=i+l;
        m[i][j]=mmin(i,j);
    }
}
```

```
int dpexpr()
{
    return(m[i][k] + m[k+1][j] + r[i-1] * r[k] * r[j]);
}
```

```
int mmin(int i, int j)
{
    for(k=i+1;k<j;k++) mi=min(mi,dpexpr(i,k,j));
    return(mi);
}
```



## Matrix multiplication – DP example

$$M = M_1 \cdot M_2 \cdot M_3 \cdot M_4$$

$$r_i: 10, 20, 50, 1, 100$$

$m_{1,1} = 0$	$m_{2,2} = 0$	$m_{3,3} = 0$	$m_{4,4} = 0$
$m_{1,2} = 10000$	$m_{2,3} = 1000$	$m_{3,4} = 5000$	-
$m_{1,3} = 1200$	$m_{2,4} = 3000$	-	-
$m_{1,4} = 2200$	-	-	-

# Knapsack problems

Thing	1	2	...	N
Value	$v_1$	$v_2$	...	$v_n$
Weight	$w_1$	$w_2$	...	$w_n$
Taken	$x_1$	$x_2$	...	$x_n$

$$\sum_{i=1}^n x_i \cdot v_i \rightarrow \min$$

$$\sum_{i=1}^n x_i \cdot w_i \leq W$$

DP expression

$$V(w, j) = \max\{ V(w - w_j, j - 1) + v_j, V(w, j - 1) \}$$

$V(W, n) = ?$  Matrix of  $V(w, j)$ , loops on  $w$  and on  $j$ ,  $O(nW)$