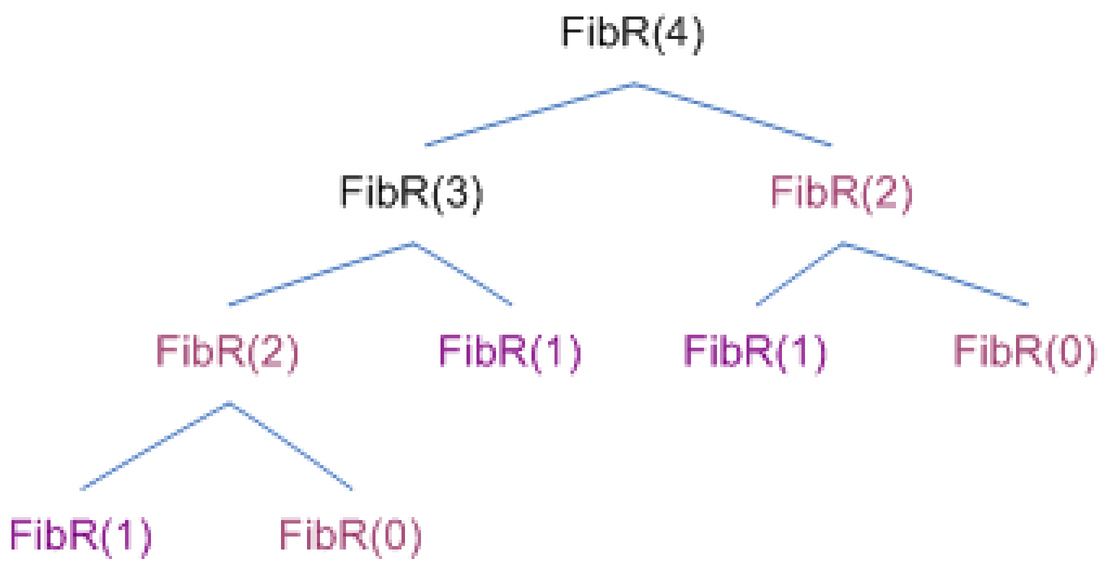
- Basic Idea
  - A simplest example: Fibonacci numbers
- Case Study
  - Longest common subsequences
  - 0/1 Knapsack
  - Weighted-interval Scheduling
  - Bellman-Ford
  - Warshall-Floyd
  - Most probable path

Basic Idea

Fibonacci numbers - recursion

```
FibR (n)
{
    if n<2 return n;
    return FibR(n-2)+FibR(n-1)
}
```

Exponential -Duplicate calls



Fibonacci numbers - iterative

```
FibI(n)
{
    int values[n+1];                    //list to store the results
    values[0] = 0;                      //former terminal values (now initial)
    values[1] = 1;
    for (i=2; i<n+1; ++i) {             //former recursion
        values[i] = values[i-2] + values[i-1]; //former main logic
    }
    return values[n];
}
```

O(n)

What is the idea of dyn-programming ?

0	1	1	2	3	5	8

Eliminates duplicate calls by

Changing directions so that iteration can be enabled

Memorizing previously calculated values

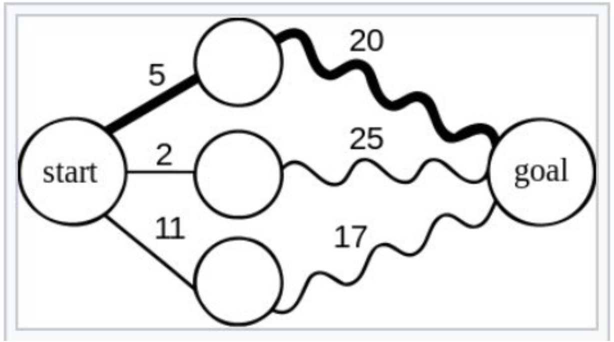
Longest Common Subsequences

Problem Statement

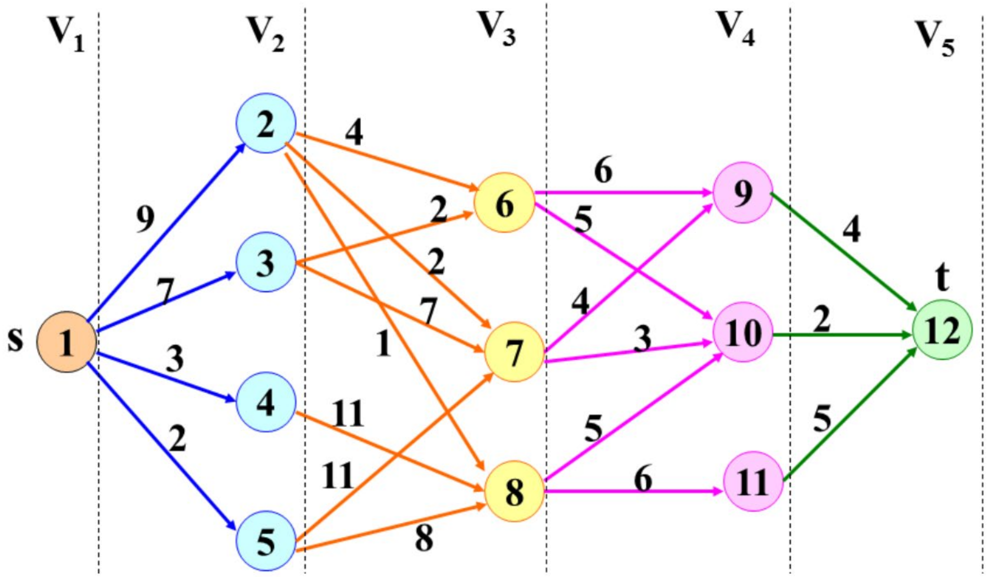
- Given 2 strings, find a longest sequence that is a subsequence of both.
  - subsequences are not required to occupy consecutive positions within the original sequences
- AxBxC
- zAzzBzzC
- LCS is ABC

Optimal substructure

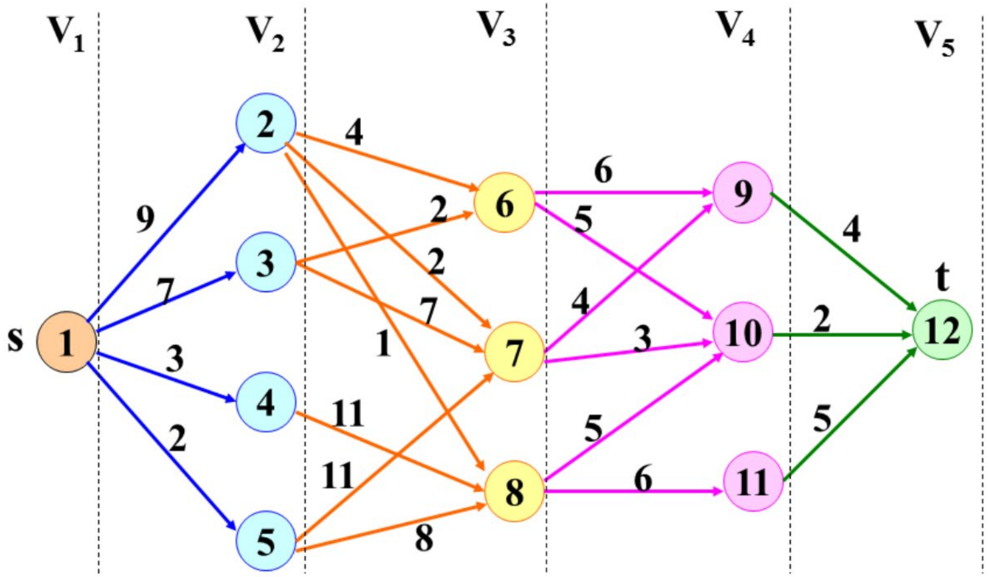
- a problem is said to have **optimal substructure** if an optimal solution can be constructed from optimal solutions of its subproblems.



(From wiki: optimal substructure)



V1
V2
V3
V4
V5
s
t
1
2
3
4
5
6
7
8
9
10
11
12
9
7
3
2
4
2
2
7
1
11
11
8
6
5
4
3
5
6
4
2
5



V_1
V_2
V_3
V_4
V_5
s
t
1
2
3
4
5
6
7
8
9
10
11
12
9
7
3
2
4
2
2
7
1
11
11
8
6
5
4
3
5
6
4
2
5

Subproblem of LCS

- String
  - Prefix
  - Suffix
  - Arbitrary

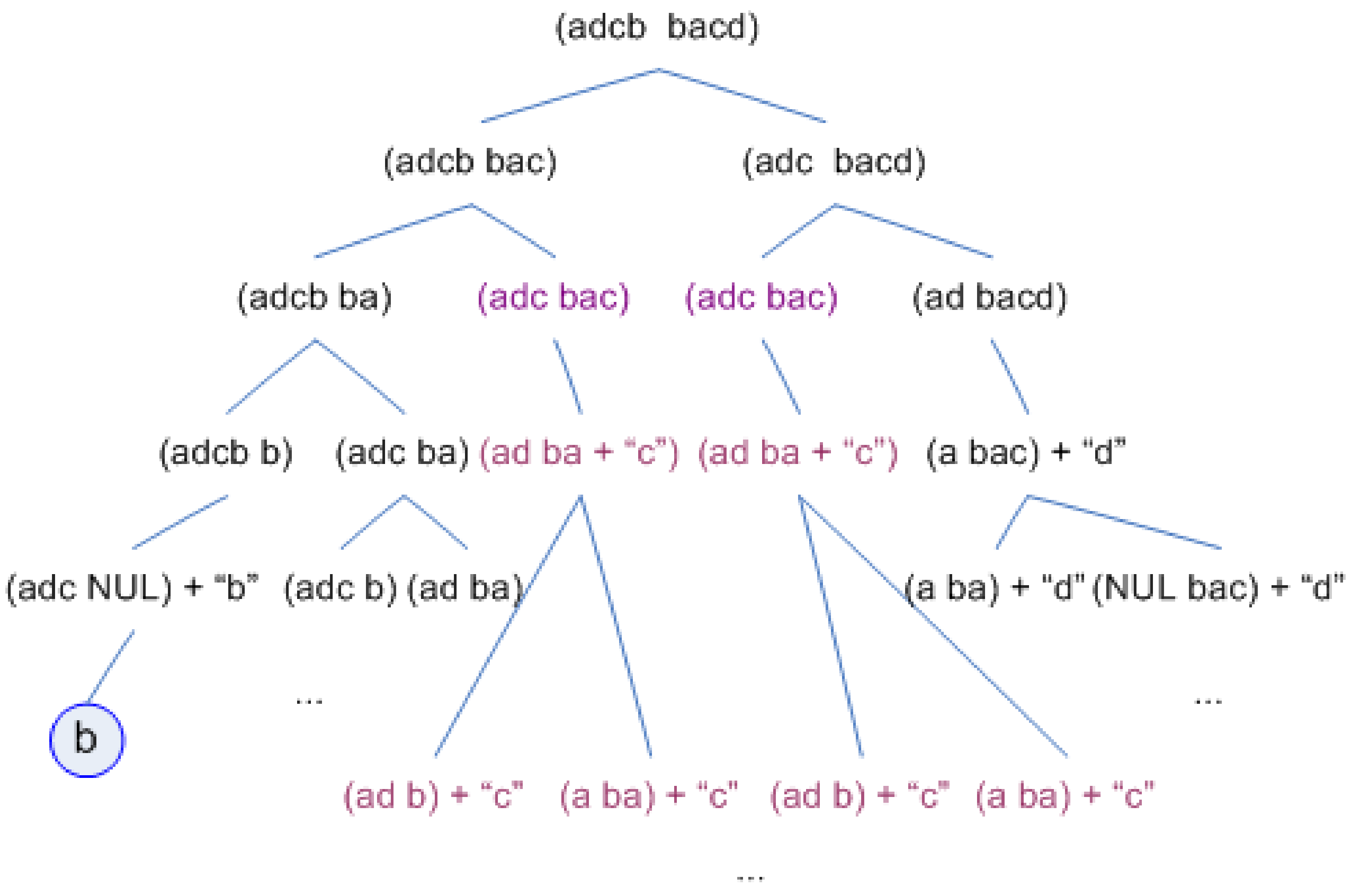
- LCS(AxBxC, AyyyyBC) equivalent to
  - LCS(AxBx,AyyyyB) + 1
  - Case 1: Prefix end in the same character

- LCS(AxBx,AyyyyB) equivalent to
  - Longest(LCS(AxBx, Ayyyy), LCS(AXB, AyyyyB))
  - Case2: Prefixs don’t have a common end character

Recursion Algorithm

```
LCS_R(s1,s2) {
    i = s1.size()-1; //last index in the first string
    j = s2.size()-1; //last index in the second string
    if ( i==0 or j==0 ) return 0;
    if ( s1[i] == s2[j] ) return 1+LCS_R(s1[0..i-1],s2[0..j-1]);
    return longest( LCS_R(s1[0..i-1],s2[0..j]) ,
                    LCS_R(s1[0..i],s2[0..j-1]) );
}
```

Duplicate calls



Iterative Algorithm

Initial

	Ø	a	d	c	b
Ø	Ø	Ø	Ø	Ø	Ø
b	Ø				
a	Ø				
c	Ø				
d	Ø				

O(|s1||s2|)

```
if ( s1[i] == s2[j] )
    values[i][j] = 1 + values[i-1][j-1]);
else
    values[i][j] = max( values[i-1][j], values[i][j-1] );
```

	Ø	a	d	c	b
Ø	Ø	Ø	Ø	Ø	Ø
b	Ø	Ø	Ø	Ø	1
a	Ø	1	1	1	1
c	Ø	1	1	2	2
d	Ø	1	2	2	2

Iterative Algorithm - traceback

	Ø	a	d	c	b
Ø	Ø	Ø	Ø	Ø	Ø
b	Ø	Ø	Ø	Ø	1
a	Ø	↖1	←1	1	1
c	Ø	↑ 1	1	↖2	←2
d	Ø	1	↖2	←↑2	←↑ 2

0/1 knapsack

Problem Statement

- M is a set of m items with values $\{v_1, v_2, \cdots v_m\}$ and weights $\{w_1, w_2, \cdots w_m\}$
- A bag with maximum weight W

- Load k(k<=m) items into the bag, such that the total weight <= W, and the total value is as large as possible

Example

- 5 items, i.e. m = 5
- With a bag, W = 10
- Values {6 3 5 4 6}
- Weight {2 2 6 5 4}

- Consider the last item
  - Put it into the bag
  - The value of original problem (5 items) turns to 6+optimal(4 items and a bag with weight 10-4 = 6)

  - Not put it into the bag
  - The value of original problem turns to 0+optimal(4 items and a bag with weight 10)

Optimal substructure – 5 items

- For each item, select or not select

- KS(m, W) denotes the original problem
  - the optimal solution of m items and a bag with W.

Considering 5-th item:
v_5: 6
w_5: 4

- KS(5, 10) = $\max \begin{cases} 6 + KS(4,6) & 6+9 \\ 0 + KS(4,10) & 0+14 \end{cases}$
  15

Optimal substructure – 4 items

- KS(4, 6) = $\max\begin{cases} 4+KS(3,1) & 4+0 \\ 0+KS(3,6) & 0+9 \end{cases}$

  9

- KS(4, 10) = $\max\begin{cases} 4+KS(3,5) & 4+9 \\ 0+KS(3,10) & 0+14 \end{cases}$

  14

Considering 4-th item:

v_4: 4

w_4: 5

Optimal substructure – 3 items

- KS(3, 1) = KS(2,1) 0

- KS(3, 5) = KS(2,5) 9

Considering 3-rd item:
v_3: 5
w_3: 6

- KS(3,6) = $\max \begin{cases} 5 + KS(2,0) & 5+0 \\ 0 + KS(2,6) & 0+9 \end{cases}$
9

- KS(3,10) = $\max \begin{cases} 5 + KS(2,4) & 5+9 \\ 0 + KS(2,10) & 0+9 \end{cases}$
14

Optimal substructure – 2 items

Considering 2-nd item:
v_2: 3
w_2: 2

- KS(2,0) = KS(1,0) 0
- KS(2,1) = KS(1,1) 0
- KS(2,4) = $\underset{9}{\max}\begin{cases}3+KS(1,2)\\0+KS(1,4)\end{cases}$
- KS(2,5) = $\underset{9}{\max}\begin{cases}3+KS(1,3)\\0+KS(1,5)\end{cases}$
- KS(2,6) = $\underset{9}{\max}\begin{cases}3+KS(1,4)\\0+KS(1,6)\end{cases}$
- KS(2,10) = $\underset{9}{\max}\begin{cases}3+KS(1,8)\\0+KS(1,10)\end{cases}$

Optimal substructure – 1 item

Considering 1-st item:
v_1: 6
w_1: 2

- KS(1,0) = KS(0,0) = 0
- KS(1,1) = KS(0,1) = 0
- KS(1,2) = $\max\begin{cases} 6 + KS(0,0) \\ 0 + KS(0,2) \end{cases}$=6
- KS(1,3) = ⋯ = 6
- KS(1,4) = ⋯ = 6
- KS(1,5) = ⋯ = 6
- KS(1,6) = ⋯ = 6
- KS(1,8) = ⋯ = 6
- KS(1,10) = ⋯ = 6

Optimal substructure

- KS(i, w) = $\max \begin{cases} \text{value}_i + KS(i-1, w - weighti) \\ 0 + KS(i-1, w) \end{cases}$

Data structure

W=10
Values {6 3 5 4 6}
Weight {2 2 6 5 4}

w

Item i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6								
2	0										
3	0										
4	0										
5	0										

+6 max

+6 max

KS(5,10)

if (w >= weight_i) table[i][w] = max(table[i-1][w], table[i-1][w-weight_i] + value_i;
else table[i][w] = table[i-1][w];

O(mw)

Data structure

W=10
Values {6 3 5 4 6}
Weight {2 2 6 5 4}

w

Item i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6	6
2	0	0	6	6	9	9	9	9	9	9	9
3	0	0	6	6	9	9	9	9	11	11	14
4	0	0	6	6	9	9	9	10	11	13	14
5	0	0	6	6	9	9	12	12	15	15	15

if (w >= weight_i) table[i][w] = max(table[i-1][w], table[i-1][w-weight_i] + value_i;
else table[i][w] = table[i-1][w];

O(mw)

Weighted Interval-scheduling

Problem Statement

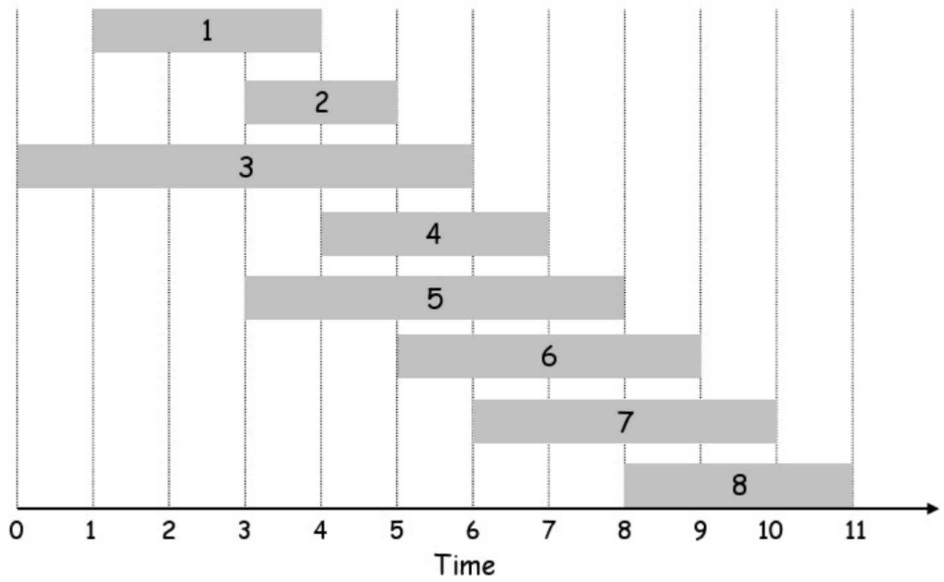
- Job j starts at s_j, finishes at f_j, and has weight/value v_j
- Two jobs compatible if they don't overlap
- Goal: find maximum weight subset of mutually compatible jobs

Input: {(3,8), (0,6), (6,10), (4,7), (1,4), (3,5), (8,11), (5,9)}

Example

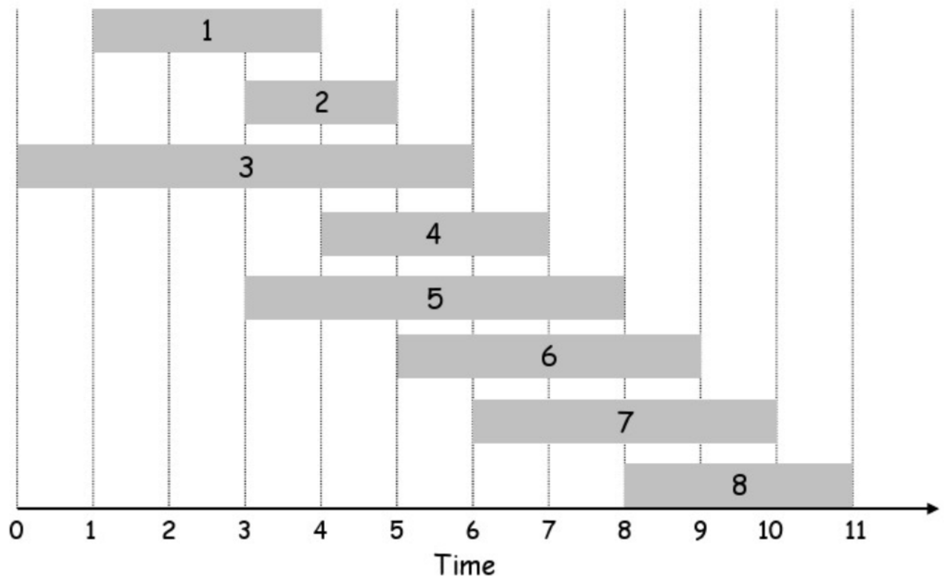
Input: {(3,8), (0,6), (6,10), (4,7), (1,4), (3,5), (8,11), (5,9)}

Sort it basing on their **finish** time



j	start	finish
1	1	4
2	3	5
3	0	6
4	4	7
5	3	8
6	5	9
7	6	10
8	8	11

Example



j	weight(j)	p(j)
0		
1	3	0
2	1	0
3	6	0
4	5	1
5	1	0
6	2	2
7	4	3
8	2	5

p(j) is the largest index i<j such that job i is compatible with j

Optimal substructure

- For each job, select or not select

$$opt(0) = 0$$

$$opt(j) = \max \begin{cases} weight_j + opt(p(j)) & select \\ opt(j-1) & not\ select \end{cases}$$

0	1	2	3	4	5	6	7	8
0	3	3	6	8	8	8	10	10

j	weight(j)	p(j)
0		
1	3	0
2	1	0
3	6	0
4	5	1
5	1	0
6	2	2
7	4	3
8	2	5

opt[j] = max(opt[j-1], weight[j] + opt[p[j]]);

Optimal Substructure

Longest common subsequences

Optimal substructure of LCS

Two strings $X = x_1x_2 \ldots x_m$ and $Y = y_1y_2 \ldots y_n$

$$LCS(X, Y) = LCS(x_1x_2 \ldots x_{m-1}, y_1y_2 \ldots y_{n-1}) + 1 \qquad \text{if } x_m == y_n$$

$$LCS(X, Y) = longer(LCS(x_1x_2 \ldots x_m, y_1y_2 \ldots y_{n-1}), LCS(x_1x_2 \ldots x_{m-1}, y_1y_2 \ldots y_n)) \qquad \text{if } x_m \mathrel{!=} y_n$$

Not all problem have optimal substructure.

We need to prove it.

What to prove …

Two strings $X = x_1x_2 \ldots x_m$ and $Y = y_1y_2 \ldots y_n$

Suppose $Z = LCS(X,Y) = z_1z_2 \ldots z_k$

case 1: if $x_m == y_n$ then $z_k == x_m == y_n$ and

$LCS(x_1x_2 \ldots x_{m-1}, y_1y_2 \ldots y_{n-1}) = z_1z_2 \ldots z_{k-1}$

case 2: if $x_m \mathrel{!}= y_n$ then $Z ==$ longer $(LCS(x_1x_2 \ldots x_m, y_1y_2 \ldots y_{n-1}),$

$LCS(x_1x_2 \ldots x_{m-1}, y_1y_2 \ldots y_n))$

What to prove

Two strings $X = x_1x_2 \ldots x_m$ and $Y = y_1y_2 \ldots y_n$

Suppose $Z = LCS(X,Y) = z_1z_2 \ldots z_k$

case 1: if $x_m == y_n$ then $z_k == x_m == y_n$ and

$LCS(x_1x_2 \ldots x_{m-1}, y_1y_2 \ldots y_{n-1}) = z_1z_2 \ldots z_{k-1}$

case 2: if $x_m != y_n$, $z_k != x_m$ $\rightarrow$ $Z = LCS(x_1x_2 \ldots x_{m-1}, y_1y_2 \ldots y_n))$

case 3: if $x_m != y_n$, $z_k != y_n$ $\rightarrow$ $Z = LCS(x_1x_2 \ldots x_m, y_1y_2 \ldots y_{n-1}))$

Prove – case 1

1) if $x_m == y_n$ $z_k == x_m == y_n$

Contradiction
We assume $z_k != x_m$, then $z_1z_2... z_kx_m$ is the common sequence of X and Y, which is longer than Z, reach the contradiction.

2) $LCS(x_1x_2...x_{m-1} , y_1y_2... y_{n-1}) = z_1z_2... z_{k-1}$

Contradiction
We assume there is a W which is also a common sequence of $(x_1x_2...x_{m-1} , y_1y_2... y_{n-1})$, and $|W| > k-1$

Consider $W+x_m$， is a common sequence longer than k. it contradict with Z is the LCS of X and Y

Prove – case 2

if $x_{m\ !=}\ y_n$ and $z_{k\ !=}\ x_m$ → $Z =$ LCS($x_1x_2 \ldots xm_{-1\ ,}\ y_1y_2 \ldots$ yn))

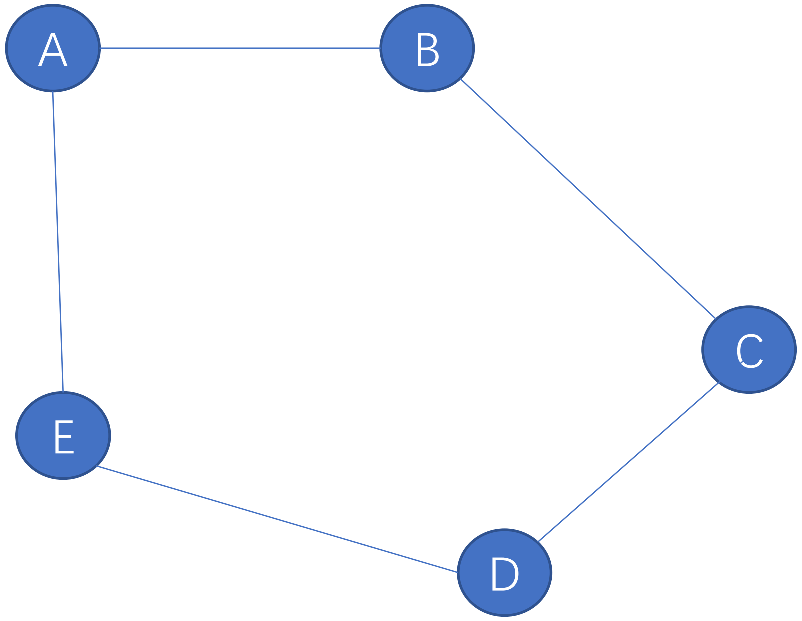
Contradiction

again we assume there is another W which is also a common sequence of ($x_1x_2 \ldots x_{m-1}$, $y_1y_2 \ldots y_n$), and |W| > k,

then W is also a common sequence of X and Y, it contradict with Z is the LCS of X and Y

Proof of case 3 is similar to case 2

problem without optimal substructure



Longest path from A to C: A-E-D-C

According to optimal substructure:

A-E-D-C == (A-E-D + D-C)

And A-E-D should be the longest path from A to D

But

The longest path form A to D is: A-B-C-D

So this is a problem has no optimal substructure

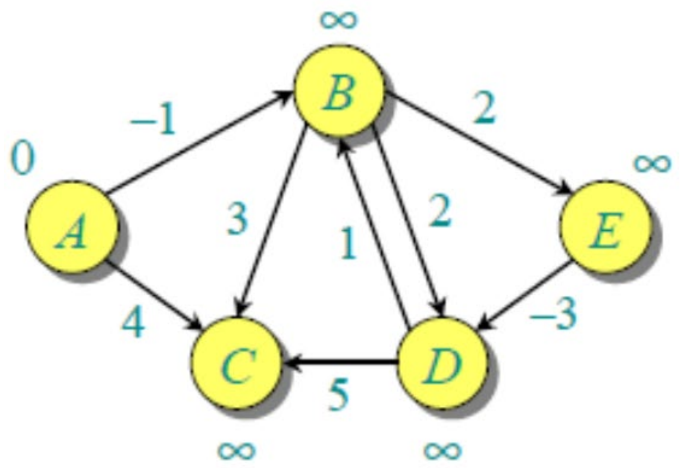
Bellman-ford

For computing shortest paths from a single source vertex to all of the other vertices in a weighted directed graph with positive or negative weights.

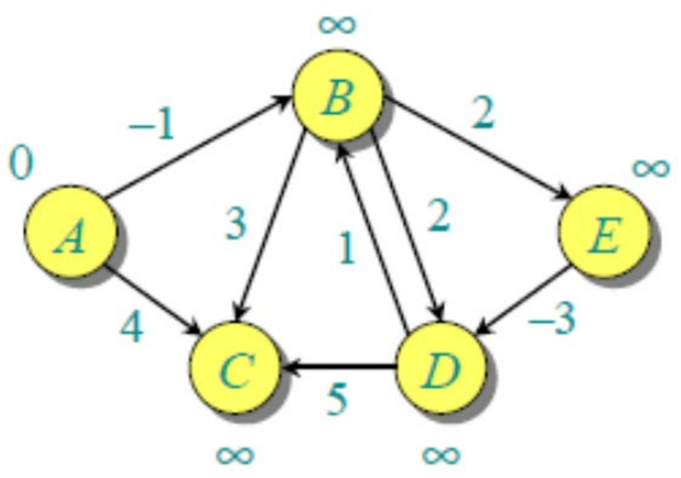
Example – ini

A directed graph with 5 vertices

Source: A



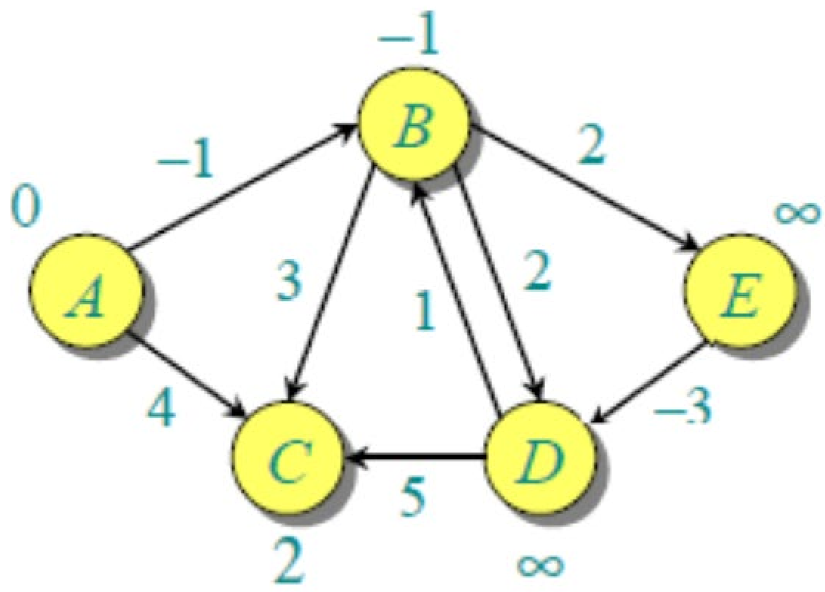
A	B	C	D	E
0	∞	∞	∞	∞



A	B	C	D	E
0	∞	∞	∞	∞

(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)

Example – first iteration

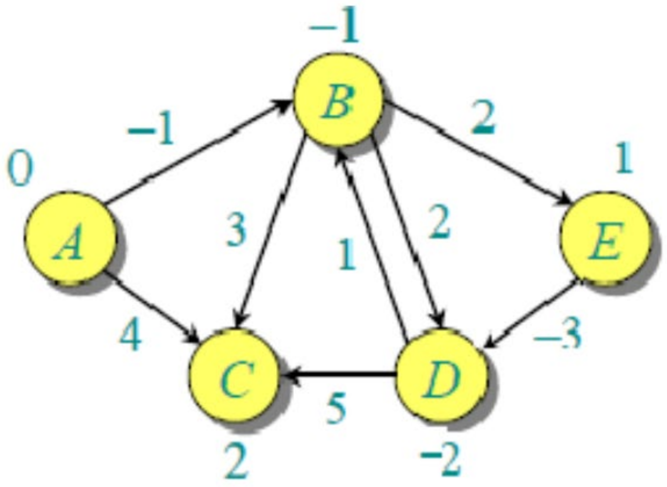


A	B	C	D	E	
0	∞	∞	∞	∞	
0	−1	∞	∞	∞	(B,E), (D,B), (B,D), (A,B)
0	−1	4	∞	∞	(A,C)
0	−1	2	∞	∞	(D,C), (B,C), (E,D)

Suppose the edge (u,v) processing sequence is: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)

```
if (d[u] + w(u,v) < d[v]) d[v] = d[u] + w(u,v);
```

Example – second iteration



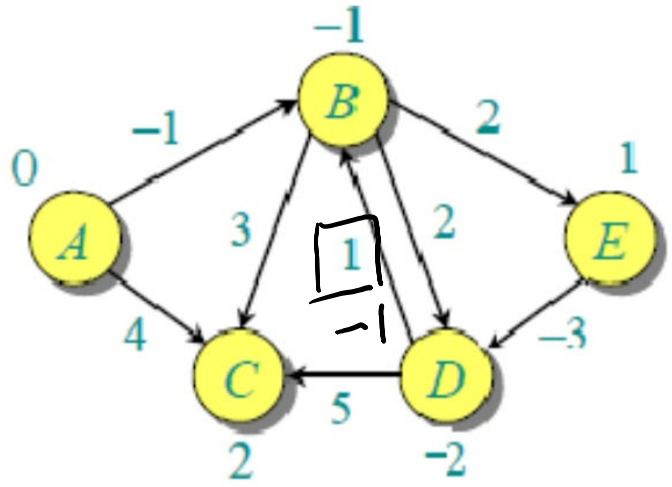
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1
0	–1	2	–2	1

Suppose the edge processing sequence is: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)

```
if (d[u] + w(u,v) < d[v]) d[v] = d[u] + w(u,v);
```

Example – … Detect negative cycles

cost(BEDB) = 2-3-1=-2.
↑
negative cycle,



A	B	C	D	E
0	∞	∞	∞	∞
0	−1	∞	∞	∞
0	−1	4	∞	∞
0	−1	2	∞	∞
0	−1	2	∞	1
0	−1	2	1	1
0	−3	2	−2	1

Suppose the edge processing sequence is: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)

```
if (d[u] + w(u,v) < d[v]) d[v] = d[u] + w(u,v);
```

Bellman-ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| - 1
        do for each edge (u, v) ∈ E[G]
                do RELAX(u, v, w)
for each edge (u, v) ∈ E[G]
        do if d[v] > d[u] + w(u, v)
                then return FALSE
return TRUE
```

O(|V||E|)

Optimal substructure

1-st iteration: from u to v through at most 0 vertex

2-nd iteration: from u to v through at most 1 vertex

...

n-1 th iteration: from u to v through at most n-2 vertices

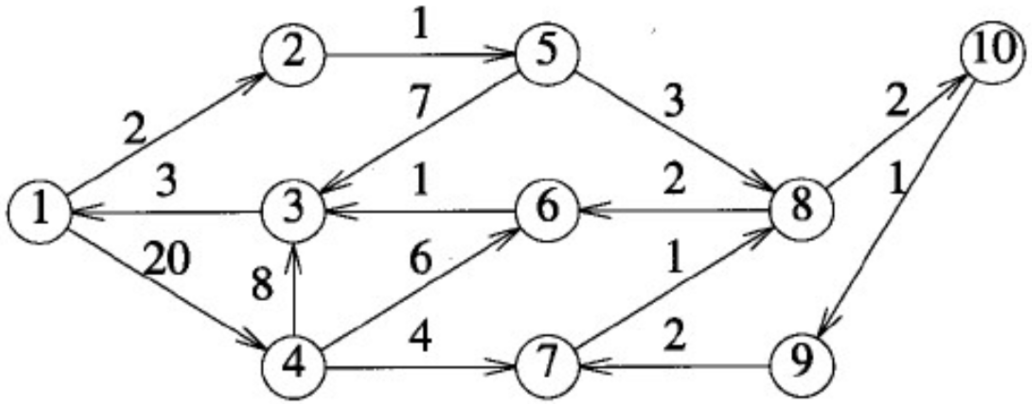
$$d[v] = \min(d[v], d[u]+w(u,v))$$

Floyd-Warshall

For finding shortest paths between *all* pairs of vertices in a weighted graph with positive or negative edge weights (but with no negative cycles)

Optimal substructure

- $C_k[i][j]$: the minimum cost of a directed path from i to j WHICH does not use nodes with indices higher than k
- $C_0[1][3]$: ∞
- $C_4[1][3]$: 28
- The shortest path from 1 to 3 is $C_{10}[1][3]$
- $C_k[i][j] \ = \ \min(C_{k-1}\,[i][j], C_{k-1}[i][k] + C_{k-1}[k][j])$



example

K=0

	1	2	3	4
1	0	6	∞	3
2	5	0	1	∞
3	3	∞	0	2
4	8	2	∞	0

K=1

	1	2	3	4
1				
2				
3				
4				

example

K=0

	1	2	3	4
1	0	6	∞	3
2	5	0	1	∞
3	3	∞	0	2
4	8	2	∞	0

K=1

	1	2	3	4
1	0	6	∞	3
2	5	0	1	8
3	3	9	0	2
4	8	2	∞	0

example

K=1

	1	2	3	4
1	0	6	∞	3
2	5	0	1	8
3	3	9	0	2
4	8	2	∞	0

K=2

	1	2	3	4
1				
2				
3				
4				

example

K=1

	1	2	3	4
1	0	6	∞	3
2	5	0	1	8
3	3	9	0	2
4	8	2	∞	0

K=2

	1	2	3	4
1	0	6	7	3
2	5	0	1	8
3	3	9	0	2
4	7	2	3	0

example

K=2

	1	2	3	4
1	0	6	7	3
2	5	0	1	8
3	3	9	0	2
4	7	2	3	0

K=3

	1	2	3	4
1				
2				
3				
4				

example

K=2

	1	2	3	4
1	0	6	7	3
2	5	0	1	8
3	3	9	0	2
4	7	2	3	0

K=3

	1	2	3	4
1	0	6	7	3
2	4	0	1	3
3	3	9	0	2
4	6	2	3	0

example

K=3

	1	2	3	4
1	0	6	7	3
2	4	0	1	3
3	3	9	0	2
4	6	2	3	0

K=4

	1	2	3	4
1				
2				
3				
4				

example

K=3

	1	2	3	4
1	0	6	7	3
2	4	0	1	3
3	3	9	0	2
4	6	2	3	0

	1	2	3	4
1	0	5	6	3
2	4	0	1	3
3	3	4	0	2
4	6	2	3	0

Algorithm

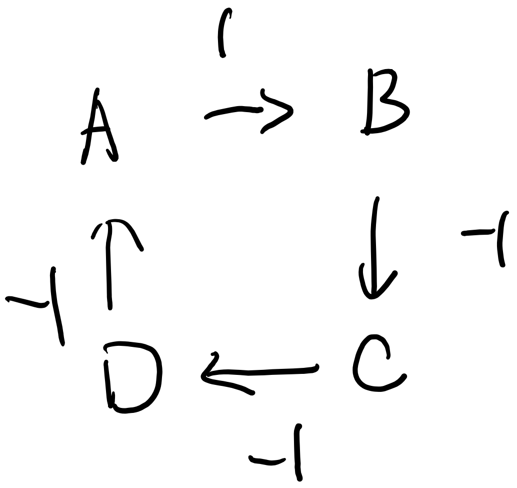
```
for(k=0;k<n;k++)
{
      for(i=0;i<n;i++)
             for(j=0;j<n;j++)
                    if(A[i][j]>(A[i][k]+A[k][j]))
                    {
                           A[i][j]=A[i][k]+A[k][j];
                           path[i][j]=k;
                     }
}
```

Path.

	1	2	3	4
1				
2				
3				
4				

$O(n^3)$

Detect Negative Cycle




```
// If distance of any verex from itself
// becomes negative, then there is a negative
// weight cycle.
for (int i = 0; i < V; i++)
    if (dist[i][i] < 0)
        return true;
return false;
}
```

	A	B	C	D
A				
B				
C				
D				

Most probable path [optional]

Viterbi

Problem Statement

```
          /                          /
left bank \ A1   A2   A3   A4 \ right bank
left bank / B1   B2   B3   B4 / right bank
left bank \ C1   C2   C3   C4 \ right bank
          /                          /
```

Find path that has the highest success probability

	A1	B1	C1
Left	1/2	1/3	1/4

Jumping layer from 4 to the right bank is always 1

	A2	B2	C2
A1	1/4	1/5	1/4
B1	1/3	1/4	1/2
C1	1	1/2	1/2

	A3	B3	C3
A2	1/2	1/4	1/2
B2	1/2	1/3	1/2
C2	1/2	1/3	2/3

	A4	B4	C4
A3	1/3	1/3	1/4
B3	1/2	2/3	1
C3	1/3	1/3	1/5

```
           /                           /
left bank \ A1    A2    A3   A4 \ right bank
left bank / B1    B2    B3   B4 / right bank
left bank \ C1    C2    C3   C4 \ right bank
           /                           /
```

	A1	B1	C1
Left	1/2	1/3	1/4

	A2	B2	C2
A1	1/4	1/5	1/4
B1	1/3	1/4	1/2
C1	1	1/2	1/2

	A3	B3	C3
A2	1/2	1/4	1/2
B2	1/2	1/3	1/2
C2	1/2	1/3	2/3

	A4	B4	C4
A3	1/3	1/3	1/4
B3	1/2	2/3	1
C3	1/3	1/3	1/5

Jumping layer from 4 to the right bank is always 1

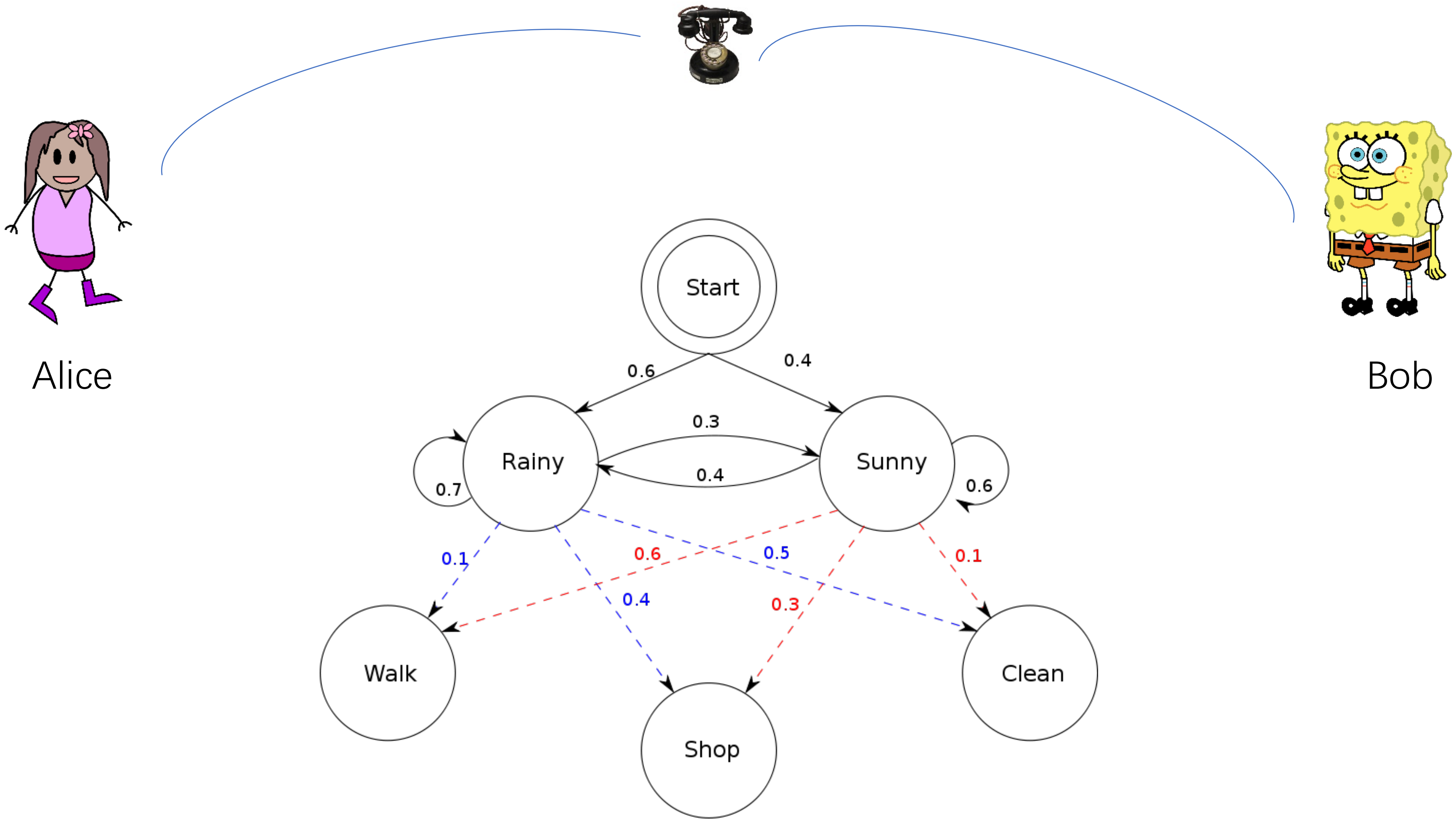
Optimal subproblem

```
                    /                     /
left bank \ A1    A2    A3   A4 \ right bank
left bank / B1    B2    B3   B4 / right bank
left bank \ C1    C2    C3   C4 \ right bank
                    /                     /
```

- 4→right bank: best= max (MPP(A4) * P(A4,right bank),
- MPP(B4) * P(B4,right bank),
- MPP(C4) * P(C4, right bank))

- $\text{Best}^{(i-1)} = \text{maxarg}_{\alpha} (\text{MPP}(\alpha) * P(\alpha, C_i))$

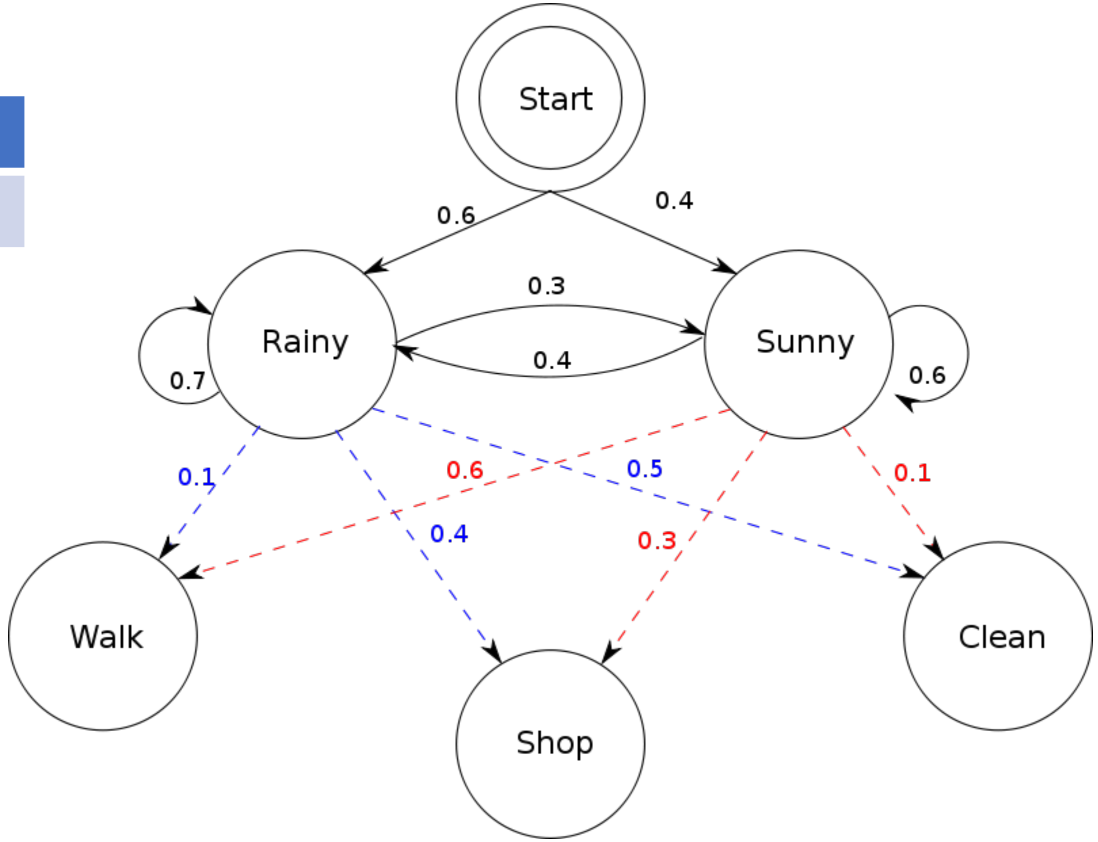
Application of Viterbi Alg.

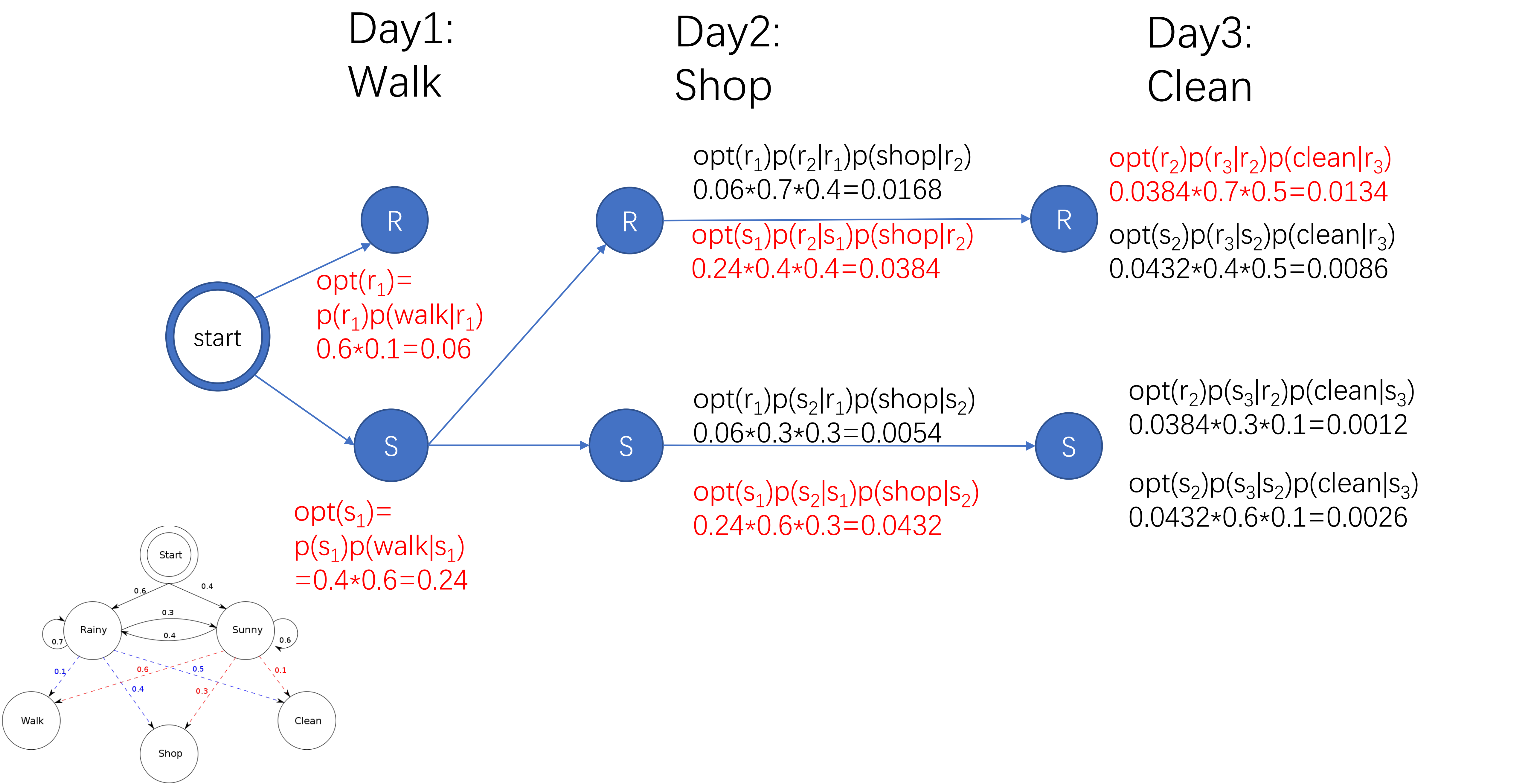


Day1	Day2	Day3
Walk	Shop	Clean

The question is:

What is the most likely weather for the three days?




Day1:
Walk
Day2:
Shop
Day3:
Clean
start
R
S
R
S
R
S
$opt(r_1)=$
$p(r_1)p(walk|r_1)$
$0.6*0.1=0.06$
$opt(s_1)=$
$p(s_1)p(walk|s_1)$
$=0.4*0.6=0.24$
$opt(r_1)p(r_2|r_1)p(shop|r_2)$
$0.06*0.7*0.4=0.0168$
$opt(s_1)p(r_2|s_1)p(shop|r_2)$
$0.24*0.4*0.4=0.0384$
$opt(r_1)p(s_2|r_1)p(shop|s_2)$
$0.06*0.3*0.3=0.0054$
$opt(s_1)p(s_2|s_1)p(shop|s_2)$
$0.24*0.6*0.3=0.0432$
$opt(r_2)p(r_3|r_2)p(clean|r_3)$
$0.0384*0.7*0.5=0.0134$
$opt(s_2)p(r_3|s_2)p(clean|r_3)$
$0.0432*0.4*0.5=0.0086$
$opt(r_2)p(s_3|r_2)p(clean|s_3)$
$0.0384*0.3*0.1=0.0012$
$opt(s_2)p(s_3|s_2)p(clean|s_3)$
$0.0432*0.6*0.1=0.0026$
Start
0.6
0.4
Rainy
Sunny
0.3
0.4
0.7
0.6
0.1
0.6
0.5
0.1
0.4
0.3
Walk
Shop
Clean