Dynamic Programming 2

Monday, April 2, 2018 5:54 PM

Last time Memoization

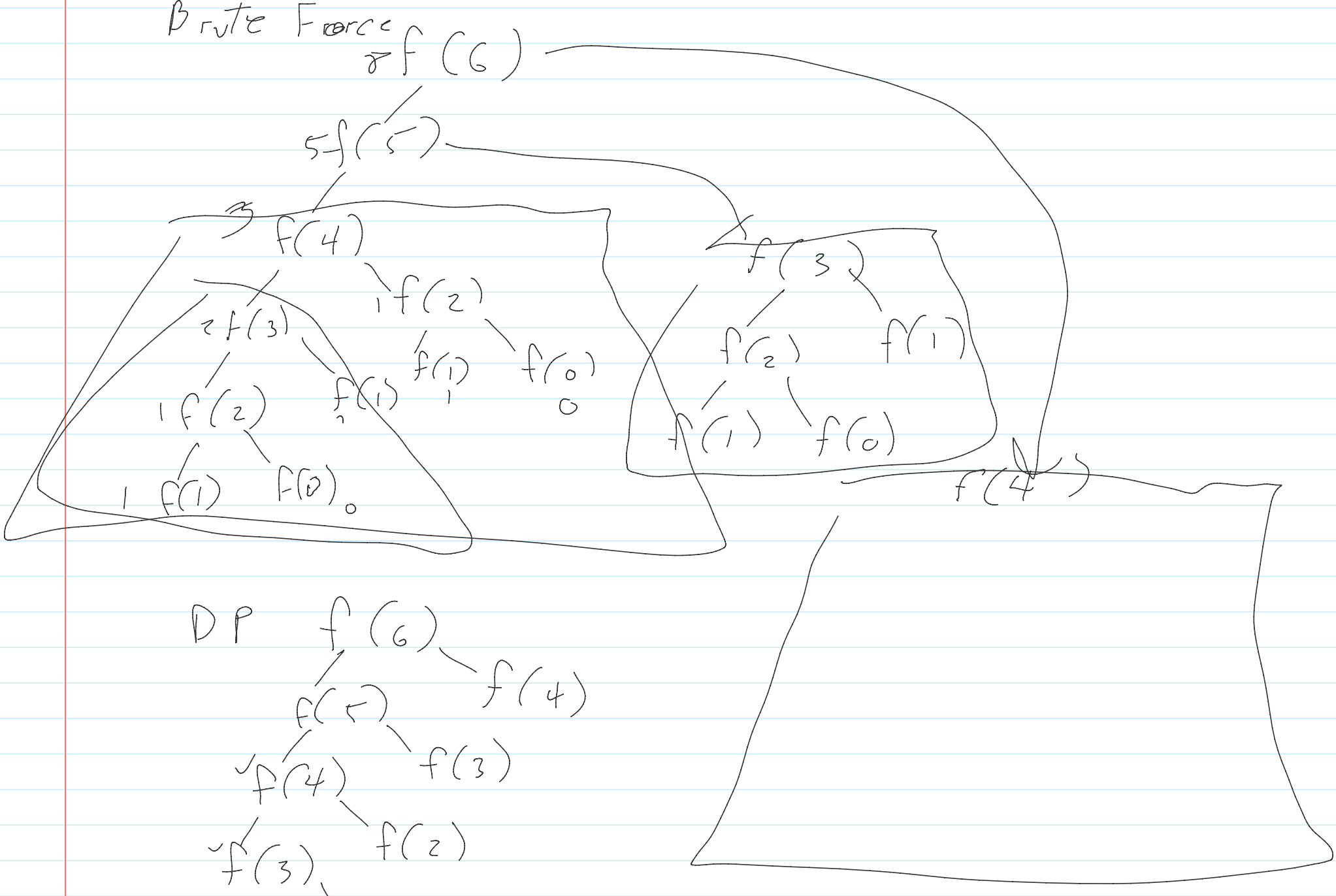
This time Bottom-up

$f(n) = f(n-1) + f(n-2)$ $f(0)=0$, $f(1)=1$

```
int f(i){
  if(i==1 || i==0) return i;

  if(memo[i] != SENT){
    return memo[i];
  }
  memo[i] = f(i-1) + f(i-2);
  return memo[i];
}
```

Brute Force



DP Bottom up

DP Bottom up

compute new(0)
new(1)

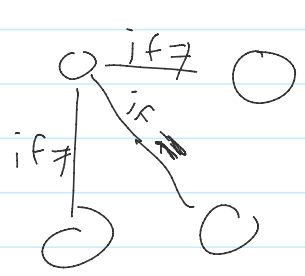
Build up the next

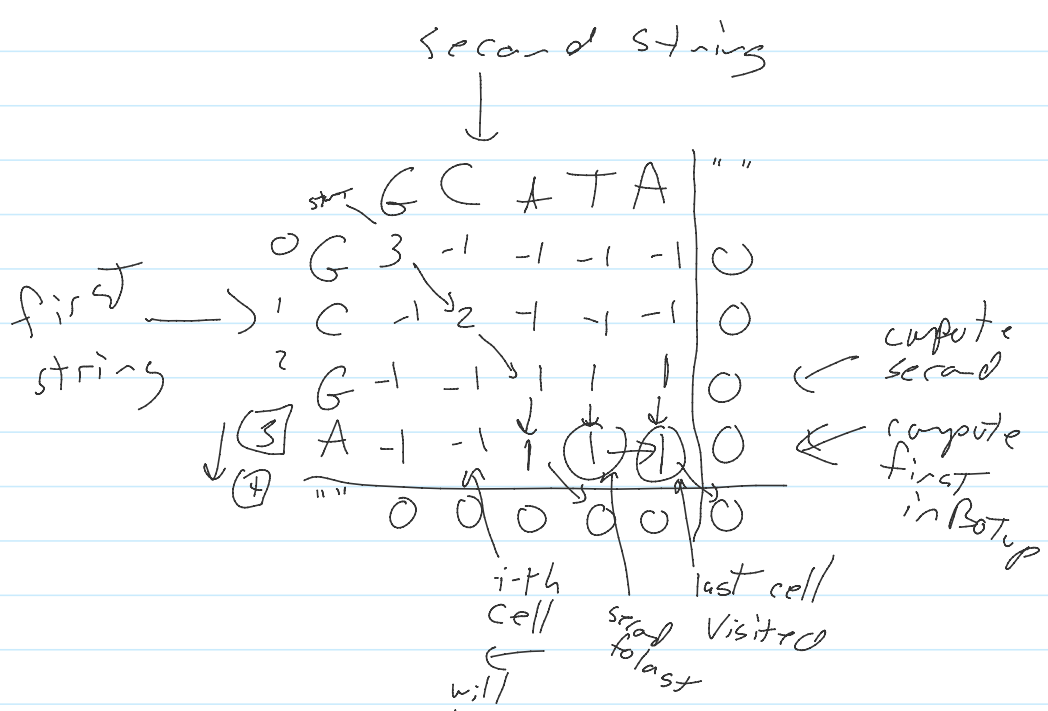
for(i=2 to n) new(i) = new(i-1) + new(i-2);

states for computation

store only necessary states
to save memory

LCS





for(int row = first string length to 1)
for(int col = second string length to 1)
compute state for row col

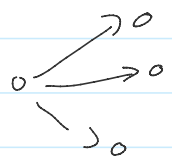
Shortest Path from a single source
to another (or all) other nodes

Effort

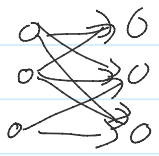
Effort

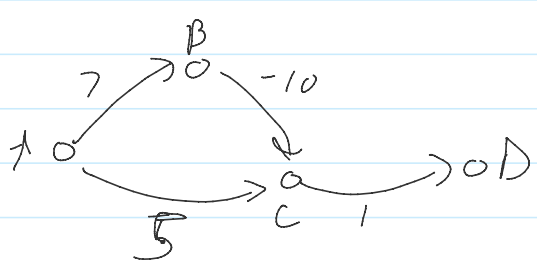
least — A* a little more efficient for single destination

more — Dijkstra's



most — Floyd's





Adjacency Matrix

From \ to

	A	B	C	D
A	0	7	5	∞
B	∞	0	-10	∞
C	∞	∞	0	1
D	∞	∞	∞	0

Use A midpoint

	A	B	C	D
A	0	7	5	∞
B	∞	0	-10	∞
C	∞	∞	0	1
D	∞	∞	∞	0

Use B

	A	B	C	D
A	0	7	-3	∞ second
B	∞	0	-10	∞ first
C	∞	∞	0	1
D	∞	∞	∞	0

Use C

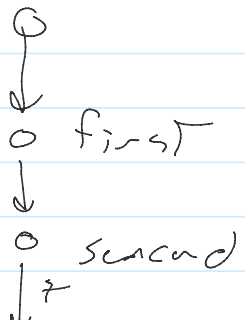
	A	B	C	D
A	0	7	-3	-2
B	∞	0	-10	-9
C	∞	∞	0	1
	∞	∞	∞	0

try all "midpoints"
for midpoint try all pairs of points
see if the pair is better off using the midpoint

Class room

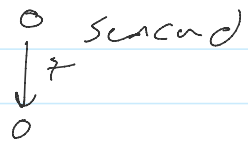
BAI atrium

Outside



Outside

Hec



Hypothetical Syllogism / Transitivity

```
for(int i=0; i<|V|;i++){
    for(int f=0; f<|V|; f++){
        for(int s=0; s<|V|; s++){
            if(dist[f][s] > dist[f][i] + dist[i][s]){
                dist[f][s] = dist[f][i] + dist[i][s];
                //build back
            }
        }
    }
}
```