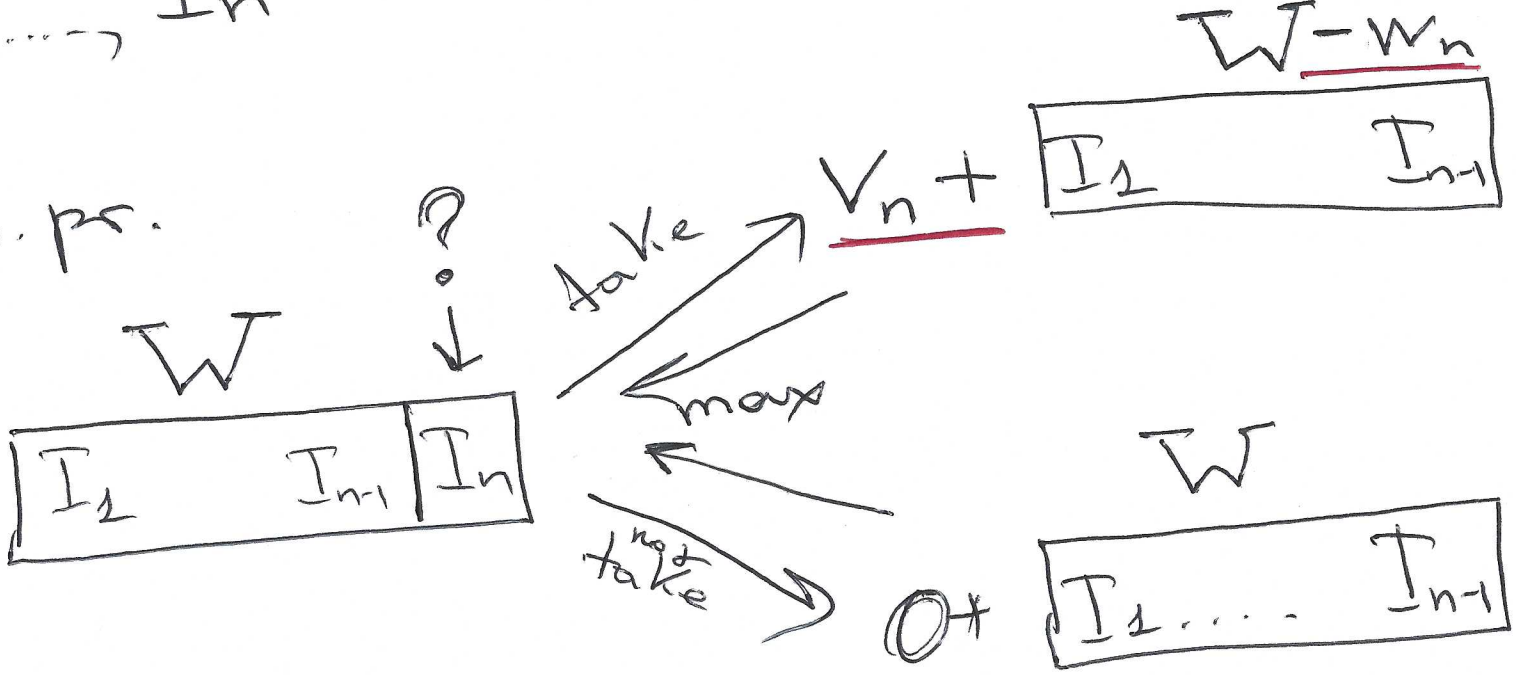
0-1 Knapsack

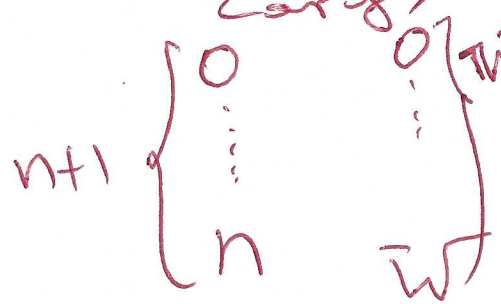
$I_1, \ldots, I_n$ ← items $\{v_i, w_i\}$

Dyn. pr.



$$KS(n, W) = \begin{cases} \max\begin{pmatrix} v_n + KS(n-1, W-w_n) \\ KS(n-1, W) \end{pmatrix} & \text{if } w_n \le W \\ 0, \quad n==0 \\ 0, \quad W==0 \end{cases}$$

2 arrays



S1 = f b a c e d

b c d a f e

LCS(f b a g b c d a f e)

	""	b	c	d	a	f	e
""	0	0	0	0	0	0	0
f	0	0	0	0	0	1	1
b	0	1	1	1	1	1	1
a	0	1	1	1	2	2	2
c	0	1	2	2	2	2	2
e	0	1	2	2	2	2	3
d	0	1	2	3	3	3	3

0

1 f

2 f b

3 f b a

4 pr1 f b a c

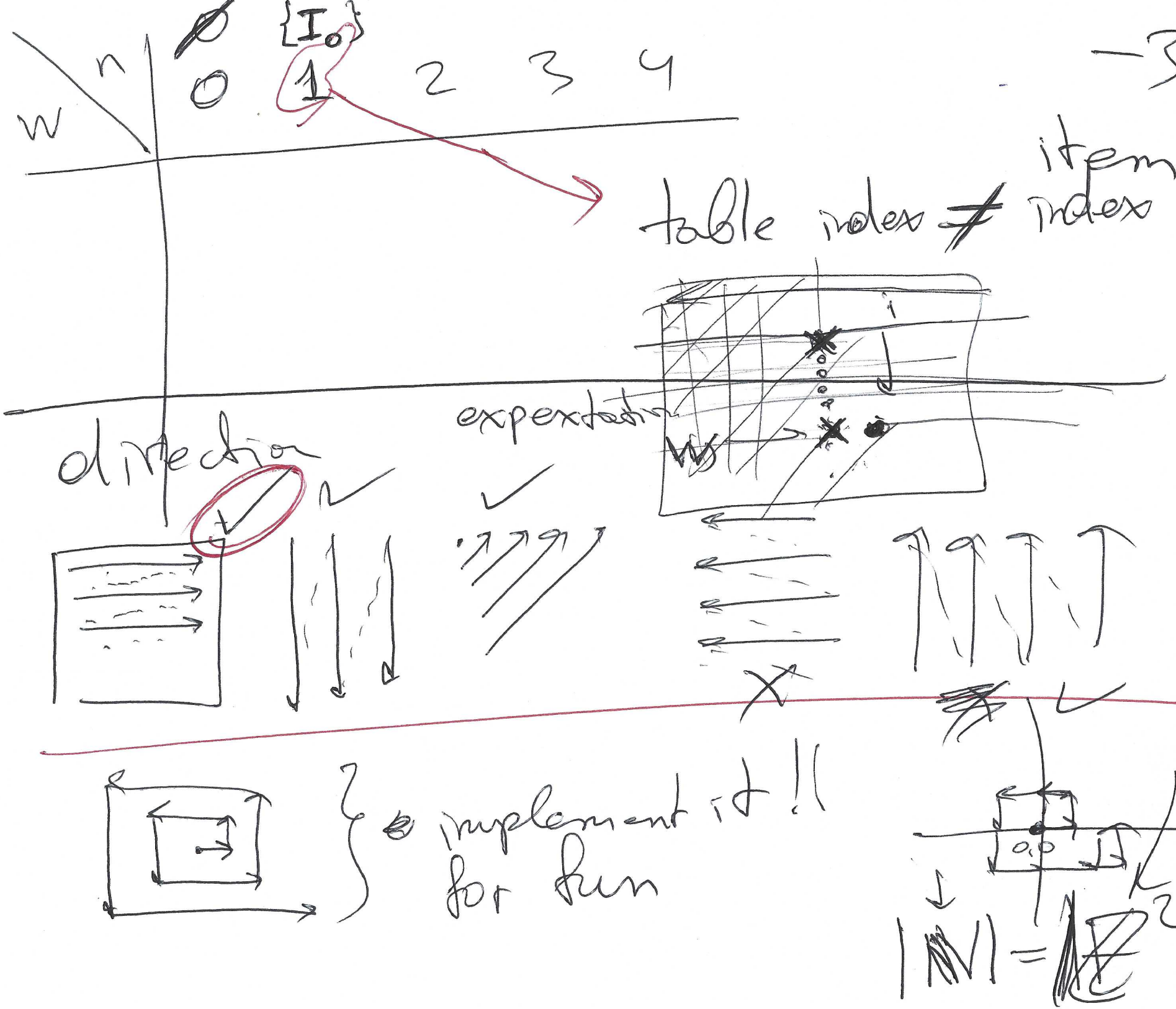
5 f b a c e

6 f b a c e d

S1

pr2

just for fun


∅
{I₀}
n
w
0
1
2
3
4
table index ≠ item index
direction
expectation
implement it !!
for fun
0,0
|N| = |E|²

```
LCS_R(s1,s2) {

    i = s1.size()-1; //last index in the first string
    j = s2.size()-1; //last index in the second string
    // check if we already have it

    if ( i==-1 or j==-1 ) return 0; // last index will be -1 for empty string
    if(t[i][j]!=-1) return t[i][j];

    if ( s1[i] == s2[j] ) return 1+LCS_R(s1[0..i-1],s2[0..j-1]);

    return t[i][j] = max( LCS_R(s1[0..i-1],s2[0..j]) , LCS_R(s1[0..i],s2[0..j-1])
);
}
```

record

record / save

memoization

table from DP (global — for now)

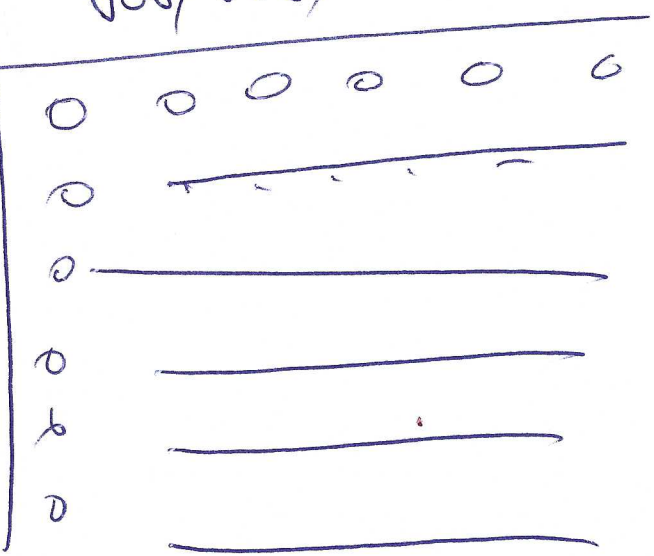
t[s1.size()+1][s2.size()+1] = { (-1,-1) };

-1 ;; "special"

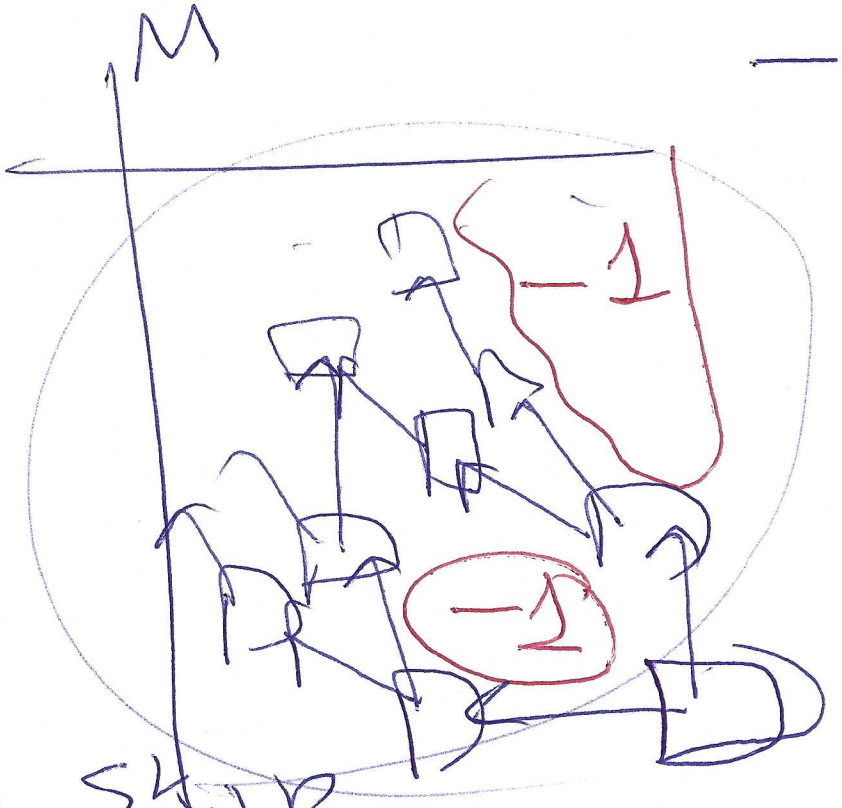
duplicates!!

DP

foo/foo/

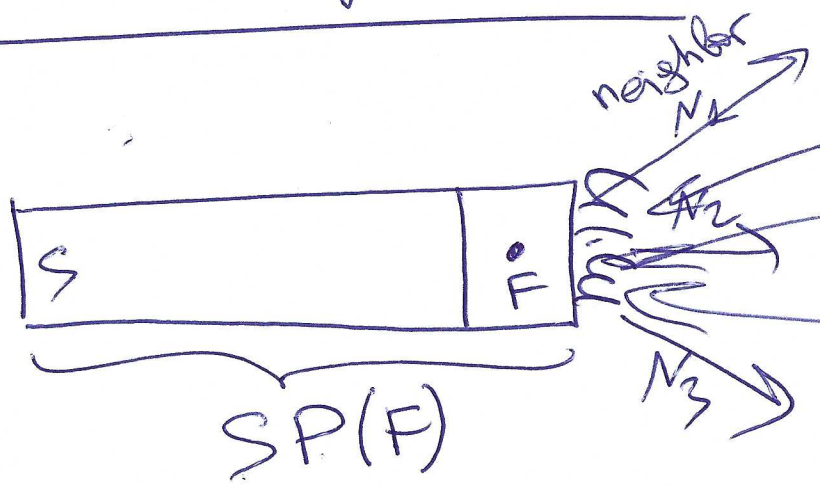


M



	FF	skip
DP	~~+~~ (Good)	does not LCS → DP
M	~~+~~ (Bad clone)	skip KS → ~~DP~~ M

Shortest path

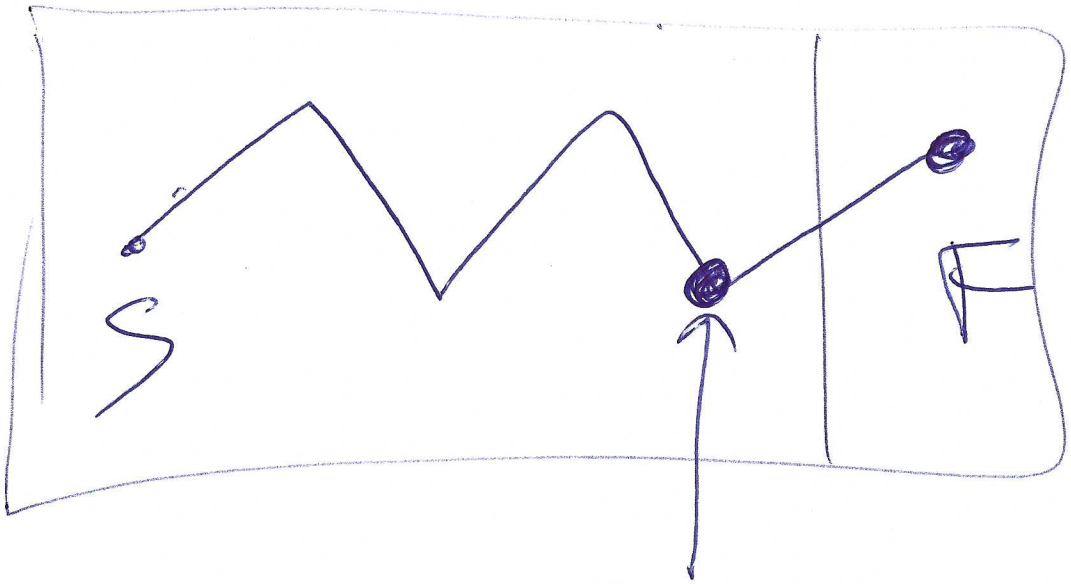


$SP \leftarrow$

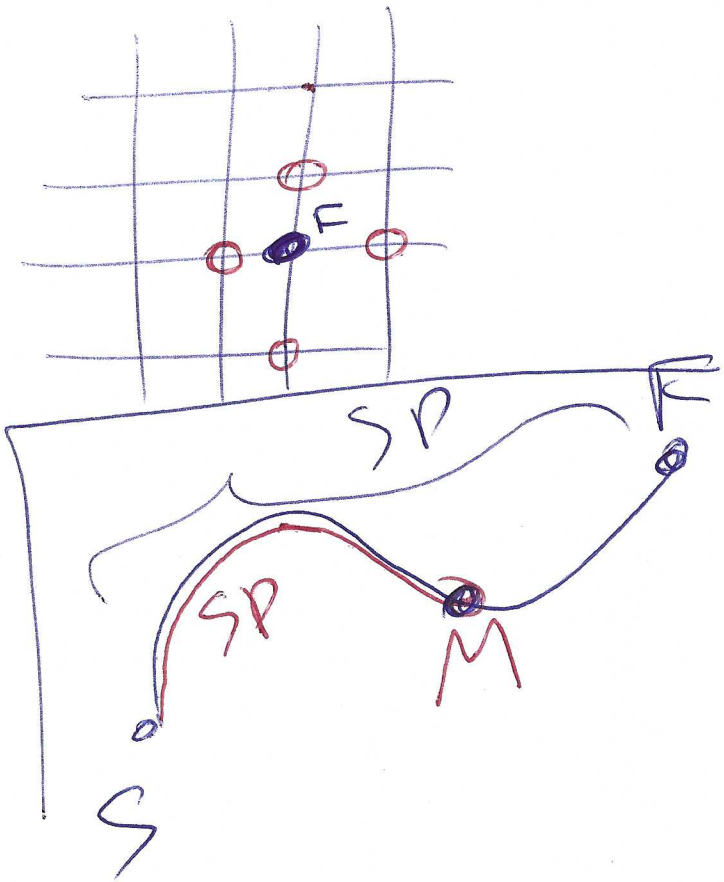
$S \dots N_1 \rightarrow F$, $SP(N_1) + d(N_1, F)$

$S \dots N_2 \rightarrow F$, $SP(N_2) + d(N_2, F)$

$S \dots N_3 \rightarrow F$, $SP(N_3) + d(N_3, F)$



Cases previous before F neighbor



fixed s

$$SP(F) = \min_{N \in Neighbors(F)} \left(SP(N) + d(N,F) \right) \quad (BF)$$

$N \to F$

alt = d[N] + len(e(N,F))

opt. if (alt < d[F])

d[F] = alt ;

Dijkstra ??

done = evaluated

w/out evaluated

— Bellman-Ford (SSSP) 1 → 5

BF allows neg. weights, for edges.

D requir $w \geq 0$.

BF is more generic than D.