General DP Remarks

Optimal Substructure

- Create optimal solution to problem using optimal solutions to subprob-
- Can't use DP if optimal solution to a problem does not require subproblem solutions to be optimal.
 - \rightarrow Often happens when subproblems are *not independent* of each other.

Overlapping Subproblems

- For DP to be useful, recursive algorithm should require us to compute optimal solutions to the same subproblems over and over again.
- In total, there should be a small number of distinct subproblems (i.e. polynomial in input size).

LCS

$$LCS[i,j] = \begin{cases} 1 + LCS[i-1,j-1] & \text{if } x_i = y_i \\ \max(LCS[i-1,j],LCS[i,j-1]) & \text{otherwise} \end{cases}$$

		j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8
			p	r	i	n	t	i	n	g
i=0		0	0	0	0	0	0	0	0	0
i=1	s	0	0	0	0	0	0	0	0	0
i=2	р	0	1	$1\leftarrow$	1←	1←	1←	1←	1←	1←
i=3	r	0	1↑	2 [^]	2←	2←	2←	2←	2←	2←
i=4	i	0	1↑	$2\uparrow$	3	3←	3←	3←	3←	3←
i=5	n	0	1↑	$2\uparrow$	3↑	4	4←	4←	4←	4←
i=6	g	0	1↑	$2\uparrow$	3↑	4↑	4↑	4↑	4↑	5
i=7	t	0	1↑	$2\uparrow$	3↑	4↑	5	5←	5←	5←
i=8	i	0	1↑	$2\uparrow$	3	4↑	5↑	6	6←	6←
i=9	m	0	1↑	$2\uparrow$	3↑	4↑	5↑	6↑	6↑	6↑
i=10	e	0	1↑	2↑	3↑	4↑	5↑	6↑	6↑	6↑ <

OBST

$$e[i,j] = \begin{cases} 0 & \text{if } i = j-1 \\ \min_{i \leq r \leq j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \leq j \end{cases}$$

root[i, j] = root of subtree with keys k_i, \ldots, k_j for $1 \le i \le j \le n$

$$w[1,\ldots,n+1,0,\ldots,n]=$$
 sum of probabilities $w[i,i-1]=0$ for $1\leq i\leq n$ $w[i,j]=w[i,j-1]+p_j$ for $1\leq i\leq j\leq n$

Consider 5 keys with search probabilities $p_1 = 0.25, p_2 = 0.2, p_3 =$ $0.05, p_4 = 0.2, p_5 = 0.3$

$\mathbf{w} \mid 0$ 0 0.25 0.45 0.50.71.0 2 0.2 $0.25 \ 0.45 \ 0.55$ 3 $0.05 \ 0.25 \ 0.55$ 4 0.20.55 0.3 1 2 3 2 2 2 4 5 5 5

1.25

 $0.3 \quad 0.75 \quad 1.35$

0.8

0 0.25 0.65

0.2

2

3

4

5

DFS

Tree edges: T Back edges: B

Cross edges: C

discovery time /finishing time 1/16 Forward edges: F $(q [s\{v(ww)v\}s] [t\{x(zz)x\}yyt]q) (r[uu]r)$

Tree Edges Are edges in depth-first forest G_{π} . Edge (u,v) is a tree edge if v was first discovered by exploring edge (u, v)

Back Edges Are edges (u, v) connecting a vertex u to an ancestor in a depth-first tree. We consider self-loops, which may occur in directed graphs, to be back edges.

Forward Edges Are nontree edges (u, v) connecting a vertex u to a descendant v in a depth-first tree.

Cross Edges Are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between verticies in different depth-first trees.

Knapsack

$$c[i, w] = \begin{cases} 0 & \text{if } i = 0 \text{ or } w = 0 \\ c[i - 1, w] & \text{if } w_i > w \\ \max(c[i - 1, w], c[i - 1, w - w_i] + v_i) & \text{if } i > 0 \text{ and } w \ge w_i \end{cases}$$

Graphs

Handshaking Lemma

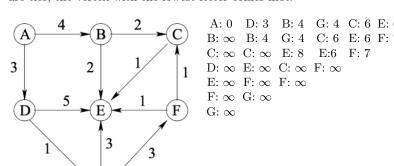
$$\sum_{v \in V} \deg(v) = 2|E|$$

BFS vs DFS

- DFS is usually for finding relationship among vertices.
- BFS Is usually for finding shortest path from a given source.

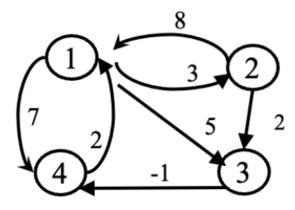
Djikstra

Execute Djikstra's algorithm on the graph below starting at A. If there are ties, the vertex with the lowest letter comes first.



Floyd-Warshall

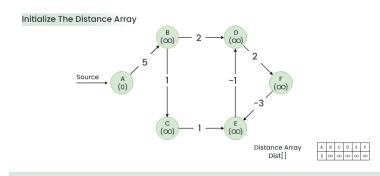
Execute the Floyd-Warshall algorithm on the graph below, provide the D^k and P matrixes on each step.



$D^{0} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$			
$D^{0} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$			
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	1		
$D^{1} = \begin{bmatrix} 4 & 2 & \infty & \infty & 0 \\ & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & \infty & \infty & 0 & -1 \\ 4 & 2 & 5 & 7 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 2 & 5 & 7 & 0 \end{bmatrix}$ $D^{2} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & 5 & 7 \\ 3 & \infty & \infty & 0 & -1 \\ 4 & 2 & 5 & 7 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 3 & \infty & \infty & 0 & -1 \\ 4 & 2 & 5 & 7 & 0 \end{bmatrix}$ $D^{3} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & 5 & 4 \\ 1 & 0 & 3 & 5 & 4 \\ 2 & 8 & 0 & 2 & 1 \\ 3 & \infty & \infty & 0 & -1 \\ 4 & 2 & 5 & 7 & 0 \end{bmatrix} \qquad P = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 2 \\ 5 \\ 7 \\ 7 \\ 0 \end{bmatrix}$ $D^{4} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 3 & 5 & 4 \\ 2 & 3 & 0 & 2 & 1 \\ 3 & 1 & 4 & 0 & -1 \end{bmatrix} \qquad P = \begin{bmatrix} 2 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \\$	$D^0 = 2$	$8 0 2 \infty$	P = 2
$D^{1} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$		∞ ∞ 0 -1	
$D^{1} = \begin{array}{ c c c c c c c c c c c c c c c c c c c$	4	$2 \infty \infty 0$	4
$D^{1} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$		1 2 3 4	
$D^{2} = \begin{bmatrix} 3 & \infty & \infty & 0 & -1 & 3 & 3 & 4 & 3 & 4 & 3 & 4 & 4 & 4 & 4$		0 3 5 7	1
$D^{2} = \begin{bmatrix} 3 & \infty & \infty & 0 & -1 & & & & 3 \\ 4 & 2 & 5 & 7 & 0 & & & 4 \end{bmatrix}$ $D^{2} = \begin{bmatrix} 1 & 2 & 3 & 4 & & & & \\ 1 & 0 & 3 & 5 & 7 & & & & 1 \\ 3 & \infty & \infty & 0 & -1 & & & 3 \\ 4 & 2 & 5 & 7 & 0 & & & 4 \end{bmatrix}$ $D^{3} = \begin{bmatrix} 1 & 2 & 3 & 4 & & & & \\ 1 & 0 & 3 & 5 & 4 & & & \\ 2 & 8 & 0 & 2 & 1 & & & P = 2 \\ 3 & \infty & \infty & 0 & -1 & & & 3 \\ 4 & 2 & 5 & 7 & 0 & & & 4 \end{bmatrix}$ $D^{4} = \begin{bmatrix} 1 & 2 & 3 & 4 & & & & \\ 1 & 0 & 3 & 5 & 4 & & & \\ 1 & 0 & 3 & 5 & 4 & & & \\ 1 & 0 & 3 & 5 & 4 & & & \\ 2 & 3 & 0 & 2 & 1 & & & P = 2 \\ 3 & 1 & 4 & 0 & -1 & & & 3 \end{bmatrix}$		8 0 2 15	P = 2
$D^{2} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$	3	I .	3
$D^{2} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$	4		4
$D^{2} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$		1 2 3 4	Ì
$D^{2} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$	1		1
$D^{3} = \begin{bmatrix} 3 & \infty & \infty & 0 & -1 & & & 3 \\ 4 & 2 & 5 & 7 & 0 & & & 4 \end{bmatrix}$ $D^{3} = \begin{bmatrix} 1 & 2 & 3 & 4 & & & \\ 1 & 0 & 3 & 5 & 4 & & \\ 3 & \infty & \infty & 0 & -1 & & & 3 \\ 4 & 2 & 5 & 7 & 0 & & & 4 \end{bmatrix}$ $D^{4} = \begin{bmatrix} 1 & 2 & 3 & 4 & & & \\ 1 & 0 & 3 & 5 & 4 & & & \\ 1 & 0 & 3 & 5 & 4 & & & \\ 2 & 3 & 0 & 2 & 1 & & & P = 2 \\ 3 & 1 & 4 & 0 & -1 & & & 3 \end{bmatrix}$	$D^2 = 2$		P = 2
$D^{3} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$	3		
$D^{3} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$			
$D^{3} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$		1 2 3 4	Ì
$D^{3} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$	1	0 3 5 4	1
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	$D^{3} = 2$	8 0 2 1	P = 2
$D^{4} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$	3		
$D^{4} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$	4	2 5 7 0	4
$D^{4} = \begin{array}{c ccccccccccccccccccccccccccccccccccc$		1 2 3 4	Ì
$D^4 = \begin{array}{c ccccc} 2 & 3 & 0 & 2 & 1 \\ & 3 & 1 & 4 & 0 & -1 \end{array} \qquad P = \begin{array}{c cccc} 2 & & & & & \\ & & & & & & \\ \end{array}$	1		1
$3 \mid 1 4 0 -1 \qquad \qquad 3 \mid$	$D^4 = 2$		P = 2
	3		3
'			
		•	ı

Bellman-Ford

Bellman-Ford is a single source shortest path algorithm that determines the shortest path between a given source vertex and every other vertex in a graph. This algorithm can be used on both weighted and unweighted graphs.



Bellman-Ford To Detect A Negative Cycle In A Graph

- 1. Initialize distance array to store shortest dist for each vertex. Initialize source as 0, and all others to be ∞ .
- 2. Relax all edges |V| 1 times.
- 3. Relax all edges one more time to detect negative cycles.

Case 1 For any edge(u, weight), if dist[u] + weight; dist[v], then there is a negative cycle.

Case 2 Case 1 fails for all edges

A	B	C	D	E	F
0	∞	∞	∞	∞	∞
0	5	∞	∞	∞	∞
0	5	6	7	∞	∞
0	5	6	7	7	9
0	5	6	6	6	9
0	5	6	5	6	8

1. Start relaxing edges, 1st relaxation:

$$Dist(B) > Dist(A) + w(A, B)$$

 $\infty > 0 + 5 \Rightarrow Dist(B) = 5$

2. 2nd relaxation:

$$Dist(D) > Dist(B) + w(B, D)$$

 $\infty > 5 + 2 \Rightarrow Dist(D) = 7$

$$Dist(C) > Dist(B) + w(B, C)$$

 $\infty > 5 + 1 \Rightarrow Dist(C) = 6$

3. 3rd relaxation:

$$Dist(F) > Dist(D) + w(D, F)$$

 $\infty > 7 + 2 \Rightarrow Dist(F) = 9$

$$Dist(E) > Dist(C) + w(C, E)$$
$$\infty > 6 + 1 \Rightarrow Dist(E) = 7$$

4. 4th relaxation:

$$Dist(D) > Dist(E) + w(E, D)$$
$$7 > 7 + (-1) \Rightarrow Dist(D) = 6$$

$$Dist(E) > Dist(F) + w(F, E)$$
$$7 > 9 + (-3) \Rightarrow Dist(E) = 6$$

5. 5th relaxation:

$$Dist(F) > Dist(D) + w(D, F)$$

 $9 > 6 + 2 \Rightarrow Dist(F) = 8$

$$Dist(D) > Dist(E) + w(E, D)$$

 $6 > 6 + (-1) \Rightarrow Dist(D) = 6$

6. 6th relaxation (final):

$$Dist(E) > Dist(F) + w(F, E)$$
$$6 > 8 + (-3) \Rightarrow Dist(E) = 5$$

$$Dist(F) > Dist(D) + w(D, F)$$

 $8 > 5 + 2 \Rightarrow Dist(F) = 7$

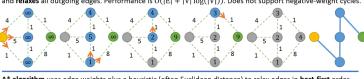
Knapsack

$$c[i,w] = \begin{cases} 0 & \text{if } i=0 \text{ or } w=0 \\ c[i-1,w] & \text{if } w_i > w \\ \max(c[i-1,w],c[i-1,w-w_i]+v_i) & \text{otherwise} \end{cases}$$

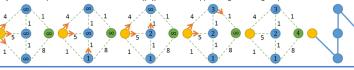
SINGLE-SOURCE SHORTEST PATH ALGORITHMS

The shortest path problem is to identify a minimum cost path in a graph from a source node to a destination.

Dijkstra's algorithm uses a priority queue to greedily select the closest vertex that has not yet been processed, and relaxes all outgoing edges. Performance is $O(|E| + |V| \log(|V|))$. Does not support negative-weight cycles



A* algorithm uses edge weights plus a heuristic (often Euclidean distance) to relax edges in best-first order. Dijkstra's is a special case. Performance is O(|E|). Does not support negative-weight cycles.

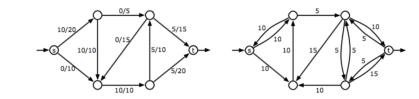


Bellman-Ford algorithm relaxes all |E| edges at most |V|-1 times. Edge selection order matters. Performance is O(|V||E|).

- = destination node
 - = node with all outgoing edges searched = edge with weight w edge in minimum cost (shortest) path = edge considered for next search

c = node (vertex) with source-to-here cost c

Network Flow



1888 BOSI

Flow Amount of material that can be transported between two nodes. **Capacity** Maximum amount of material that can be transported between two nodes.

Residual Capacity Amount of material that can still be transported between two nodes.

Augmenting Path Path from source to sink where all edges have residual capacity.

Residual Graph Graph where edges have residual capacity.