Class Meeting #18

COS 226 — Spring 2018

Mark Braverman

(based in part on slides by Kevin Wayne)

Dynamic programming

- A general algorithmic paradigm
- The go-to solution for problems with natural subproblems.
- Typically, running time polynomial but not linear (e.g Bellman-Ford is $O(E\ V)$).
- Most commonly used method for tractable combinatorial problems.
- Useful in interviews.

Dynamic programming blueprint

- Find good subproblems
- Come up with a recursive solution (recurrence relation)
- Use memorization to avoid running time blowup.

Warmup: Fibonacci numbers

F(0)=1

F(1)=1

F(n)=F(n-1)+F(n-2) for n>=2

Compute F(n)

Naïve solution

```
long F(long n) {
        if ((n==0)||(n==1)) return 1;
        return F(n-1)+F(n-2);
}
```

Running time?

Naïve solution

```
long F(long n) {
        if ((n==0)||(n==1)) return 1;
        return F(n-1)+F(n-2);
}
```

T(n) = T(n-1)+T(n-2)+1

T(n) ~ F(n) ~ ϕ^n, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.62$ ☹

Can we do better?

Replace recursion with memorization!

```
long F(long n) {
        if ((n==0)||(n==1)) return 1;
        return F(n-1)+F(n-2); }
long Fmem(long n) {
        long [] ans = new long[n+1];
        ans[0]=1; ans[1]=1;
        for (int i=2; i<=n; i++)
                ans[i]=ans[i-1]+ans[i-2];
        return ans[n];
}
```

Replace recursion with memorization!

```
long Fmem(long n) {
        long [] ans = new long[n+1];
        ans[0]=1; ans[1]=1;
        for (int i=2; i<=n; i++)
                ans[i]=ans[i-1]+ans[i-2];
        return ans[n]; }
```

Running time?

O(n)

Can do even better?

Yes, can only do O(log n) multiplications.

House coloring problem

Goal. Paint a row of n houses red, green, or blue so that

No two adjacent houses have the same color.

Minimize total cost, where cost(i, color) is cost to paint i given color.

	A	B	C	D	E	F
	7	6	7	8	9	20
	3	8	9	22	12	8
	16	10	4	2	5	7

cost to paint house i the given color

Naïve solution

Try coloring the last house on the block in all three possible colors.
$C(n)$ = the min cost of coloring the block so that the last house gets color C.

$$B(n) = cost(n, B) + \min(G(n-1), R(n-1))$$

Reduction to a smaller subproblem

Subproblem = paining the first i houses, with the last house having a prescribed color.

Can solve those and memorize the answer.

Subproblems.

R[i] = min cost to paint houses 1, …, i with i red.
G[i] = min cost to paint houses 1, …, i with i green.
B[i] = min cost to paint houses 1, …, i with i blue.
Optimal cost = min { R[n], G[n], B[n] }.

Recurrence equations:

$$R[i+1] = cost(i+1, red) + \min \{ B[i], G[i] \}$$
$$G[i+1] = cost(i+1, green) + \min \{ R[i], B[i] \}$$
$$B[i+1] = cost(i+1, blue) + \min \{ R[i], G[i] \}$$
$$R[0] = G[0] = B[0] = 0$$

Running time. O(n).

House coloring problem

	A	B	C	D	E	F
	7	6	7	8	9	20
	3	8	9	22	12	8
	16	10	4	2	5	7

cost to paint house i the given color

	A	B	C	D	E	F
R	7					
G	3					
B	16					

HOUSE COLORING PROBLEM

	A	B	C	D	E	F
	7	6	7	8	9	20
	3	8	9	22	12	8
	16	10	4	2	5	7

cost to paint house i the given color

	A	B	C	D	E	F
R	7	**9**	20	**21**	29	46
G	**3**	15	18	35	32	**34**
B	16	13	**13**	20	**26**	36

House coloring problem

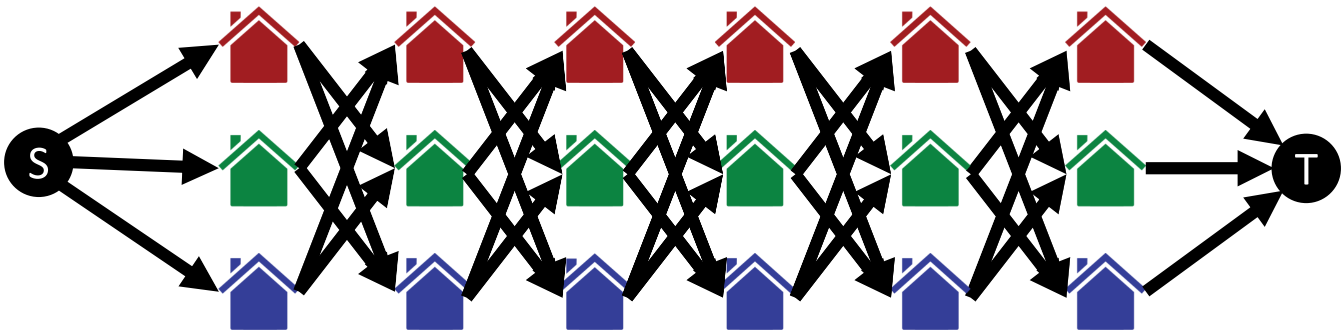
	A	B	C	D	E	F
	7	**6**	7	**8**	9	20
	3	8	9	22	12	**8**
	16	10	**4**	2	**5**	7

cost to paint house i the given color

	A	B	C	D	E	F
R	7	**9**	20	**21**	29	46
G	**3**	15	18	35	32	**34**
B	16	13	**13**	20	**26**	36

Aside: can formulate as a shortest path problem

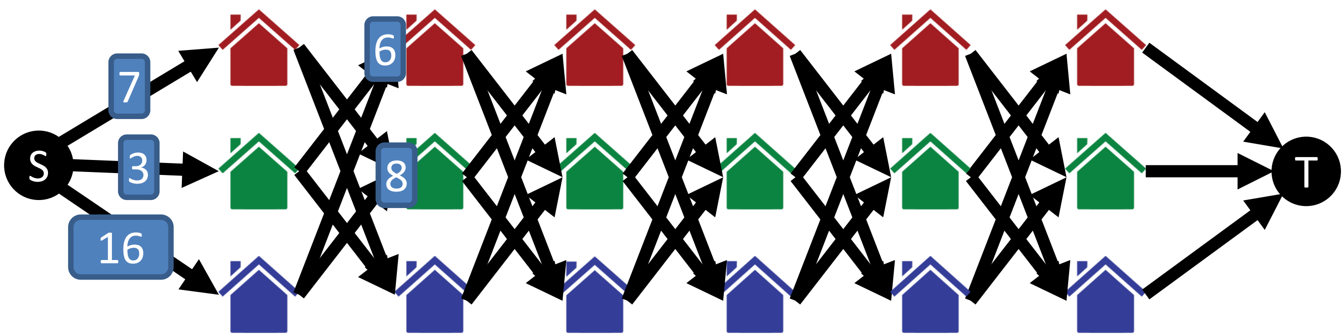
	A	B	C	D	E	F
(red)	7	6	7	8	9	20
(green)	3	8	9	22	12	8
(blue)	16	10	4	2	5	7



Aside: can formulate as a shortest path problem

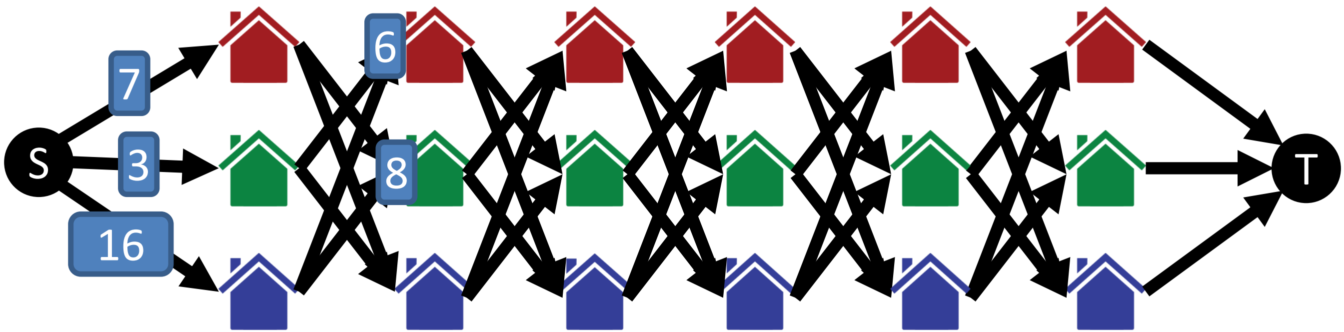
Goal: shortest path from S to T

	A	B	C	D	E	F
Red	7	6	7	8	9	20
Green	3	8	9	22	12	8
Blue	16	10	4	2	5	7



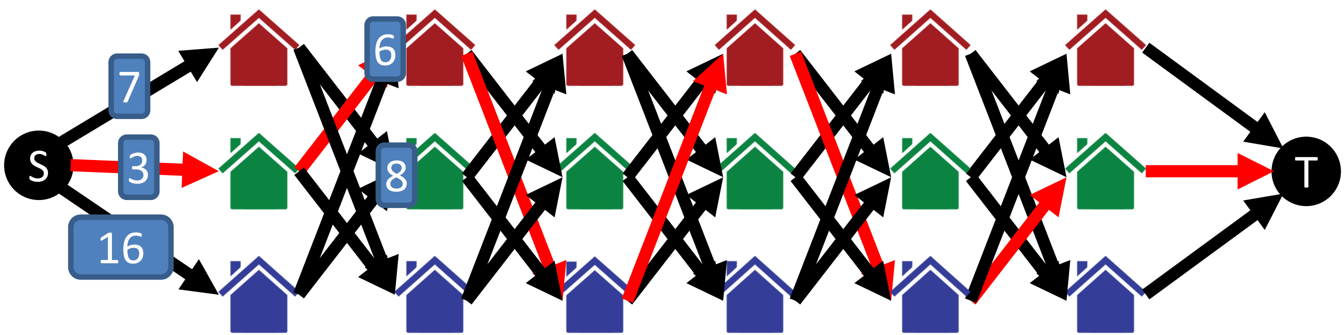
ASIDE: CAN FORMULATE AS A SHORTEST PATH PROBLEM

	A	B	C	D	E	F
R	7	**9**	20	**21**	29	46
G	**3**	15	18	35	32	**34**
B	16	13	**13**	20	**26**	36



Aside: can formulate as a shortest path problem

	A	B	C	D	E	F
R	7	**9**	20	**21**	29	46
G	**3**	15	18	35	32	**34**
B	16	13	**13**	20	**26**	36



Dynamic programming blueprint

- Find good subproblems
- Come up with a recursive solution
- Use memorization to avoid running time blowup.
- Key question: **What are the subproblems?**

COIN CHANGING

Problem. Given n coin denominations $\{ c_1, c_2, \ldots, c_n \}$ and a target value V, find the fewest coins needed to make change for V (or report impossible).

Recall. Greedy cashier's algorithm is optimal for U.S. coin denominations, but not for arbitrary coin denominations.

Ex. { 1, 10, 21, 34, 70, 100, 350, 1295, 1500 }.
Optimal. 140¢ = 70 + 70.

Def. $\mathrm{OPT}(v)$ = min number of coins to make change for v.

Goal. $\mathrm{OPT}(V)$.

Multiway choice. To compute $\mathrm{OPT}(v)$,

Select a coin of denomination c_i for some i.

Select fewest coins to make change for $v - c_i$.

Recurrence:

$$OPT(v) = \begin{cases} \infty & \text{if } v < 0 \\ 0 & \text{if } v = 0 \\ \max\limits_{1 \le i \le n} \{\, 1 + OPT(v - c_i) \,\} & \text{otherwise} \end{cases}$$

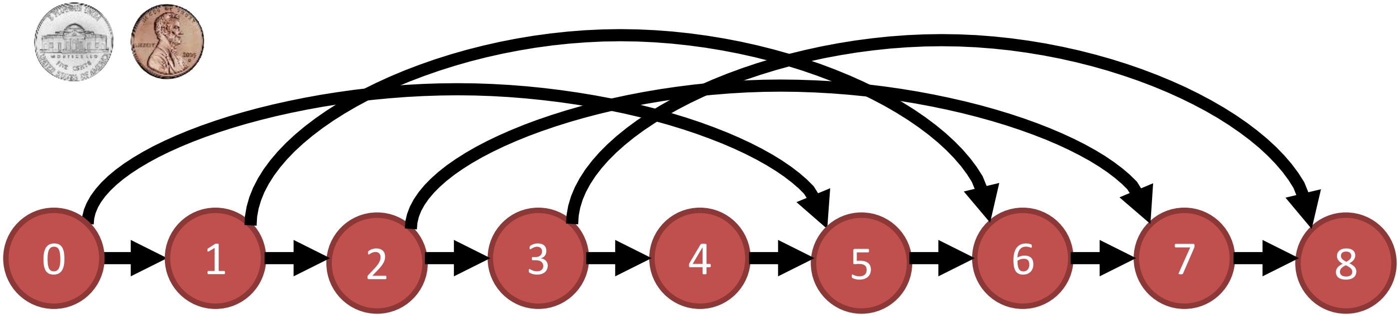
Running time. $O(n\,V)$.

Coin changing problem

Once again, can be formulated as shortest path on an appropriately chosen graph.

COIN CHANGING PROBLEM

Once again, can be formulated as shortest path on an appropriately chosen graph.

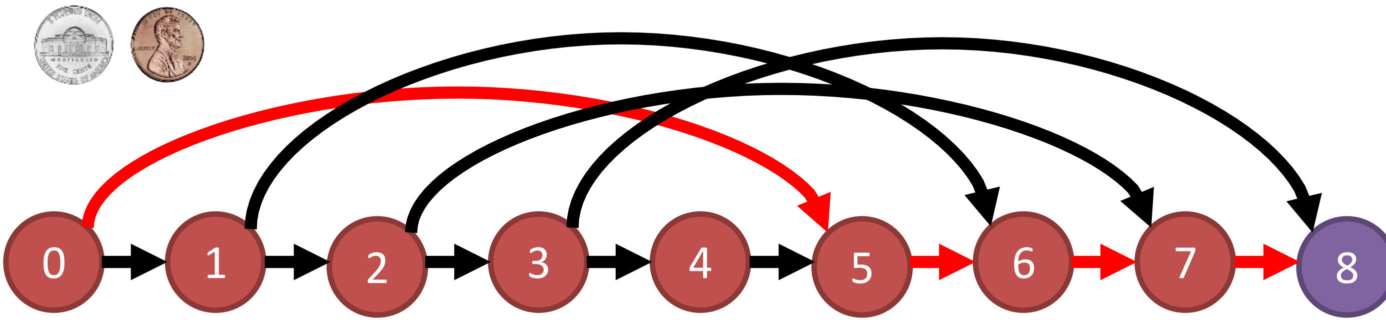


All edge weights = 1

Smallest number of coins to produce V = shortest path from 0 to V.

COIN CHANGING PROBLEM

Once again, can be formulated as shortest path on an appropriately chosen graph.



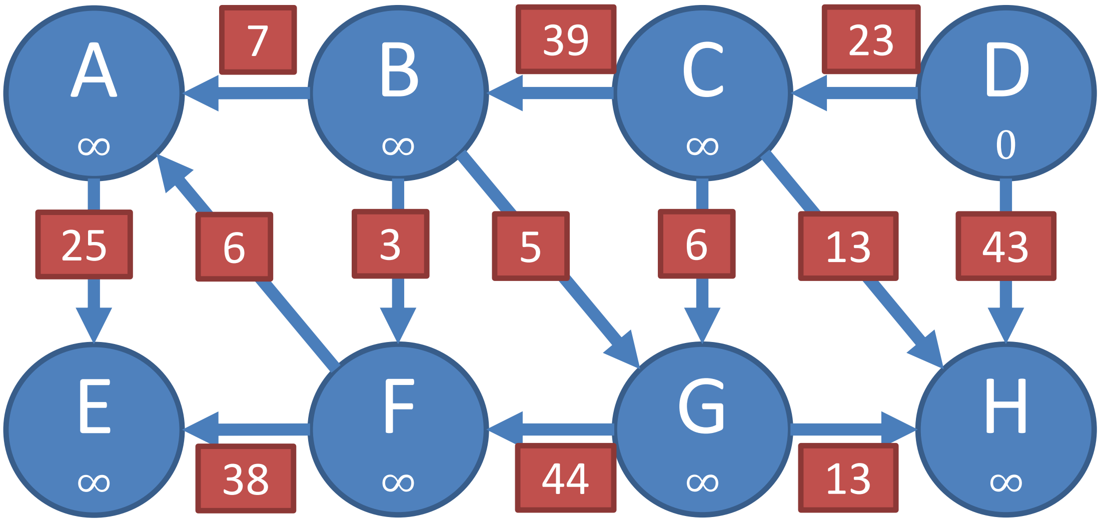
All edge weights = 1

Smallest number of coins to produce V = shortest path from 0 to V.
Is time ~n V "efficient"?

Back to graph problems

- Bellman-Ford solves the shortest path problem.
- Calculates $D(s, v)$: the distance from a given source s to all vertices v.
- Solution idea: proceed in rounds; in each round "relax" edges from each vertex in sequence.
- What is produced after k such relaxations?

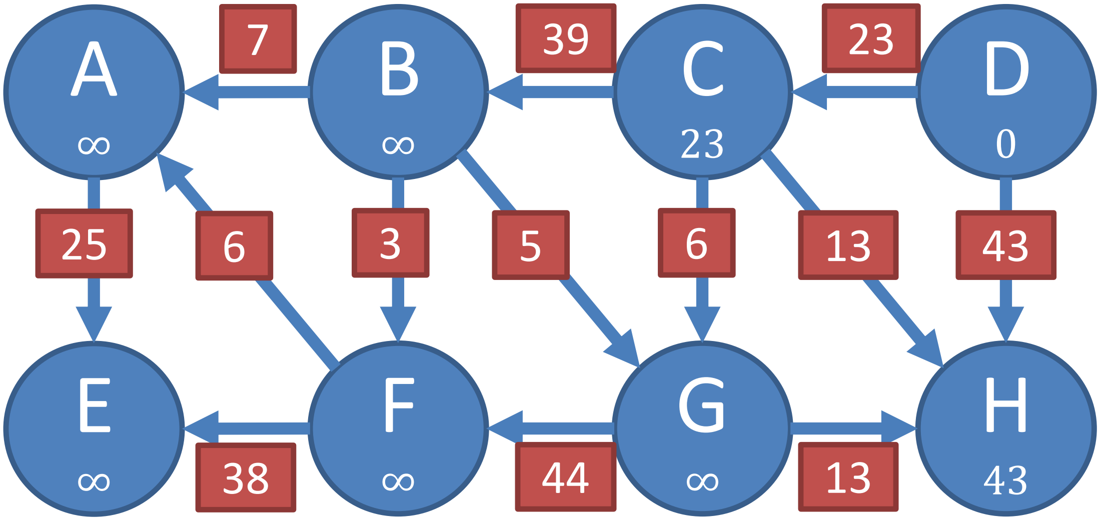
Bellman-Ford example



Relax order: A B C D E F G H

Vertex distances after 3 rounds?

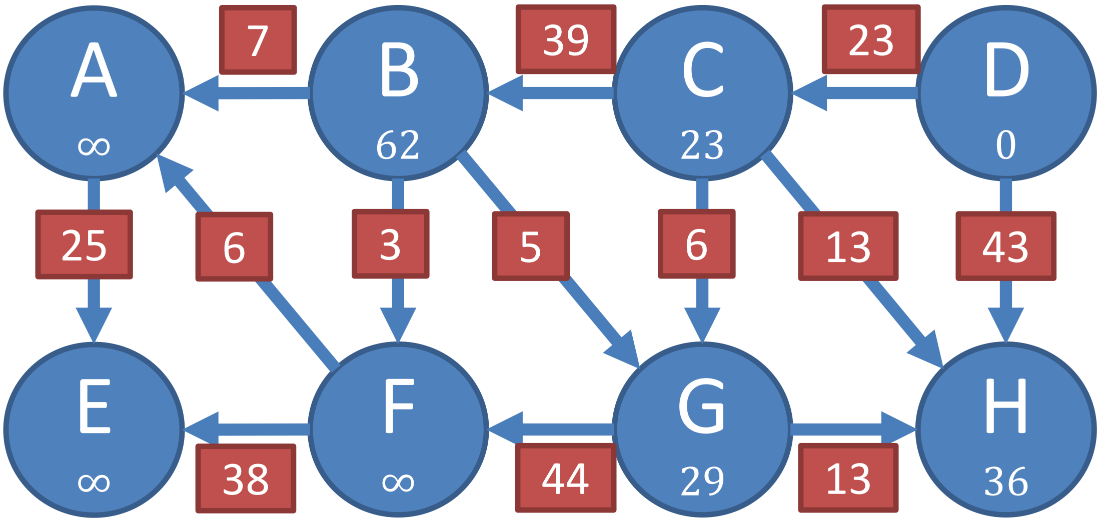
Round 1



Relax order: A B C D E F G H

D->C; D->H

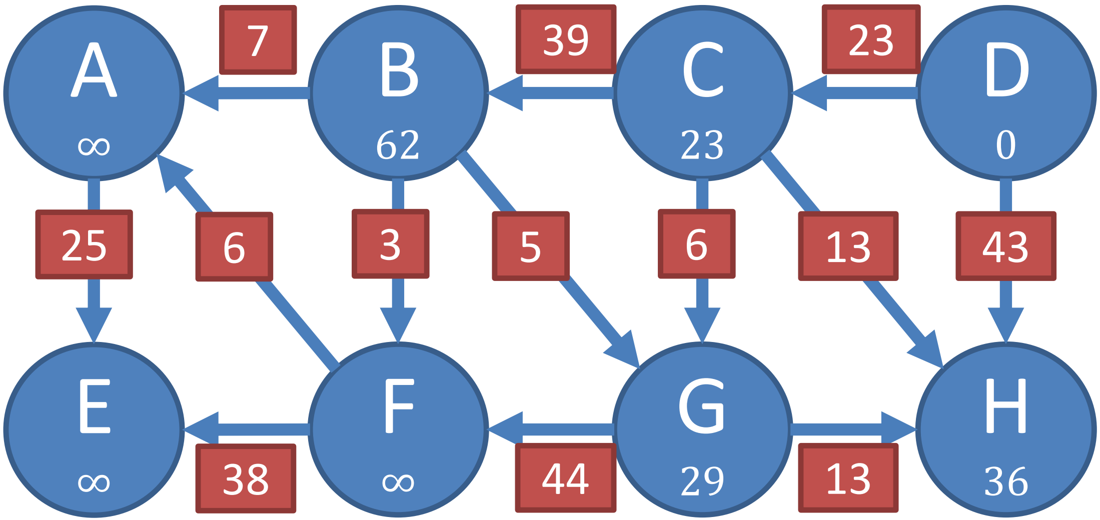
Round 2



Relax order: A B C D E F G H

C->B; C->G; C->H

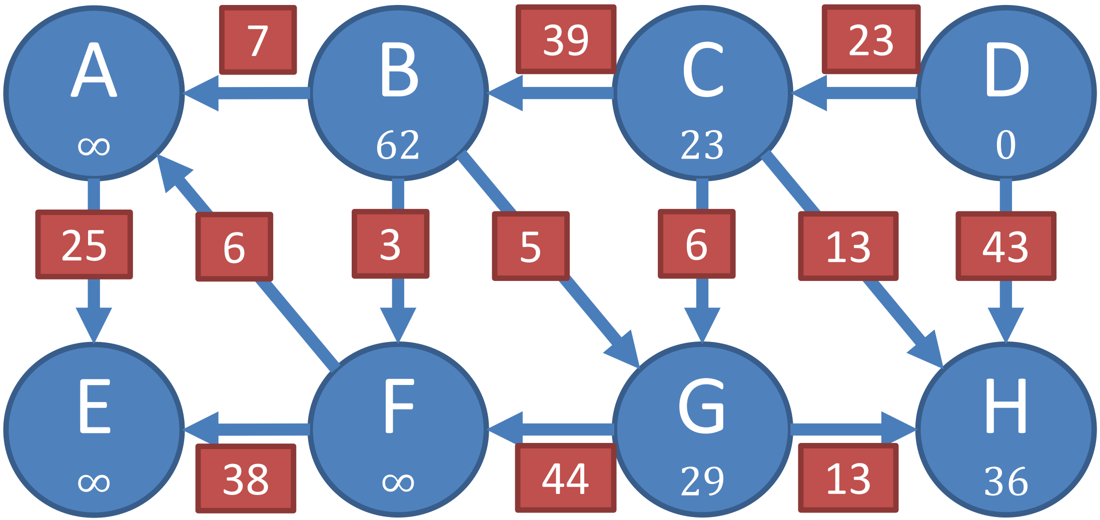
Round 2



Relax order: A B C D E F G H

C->B; C->G; C->H; D hasn't changed

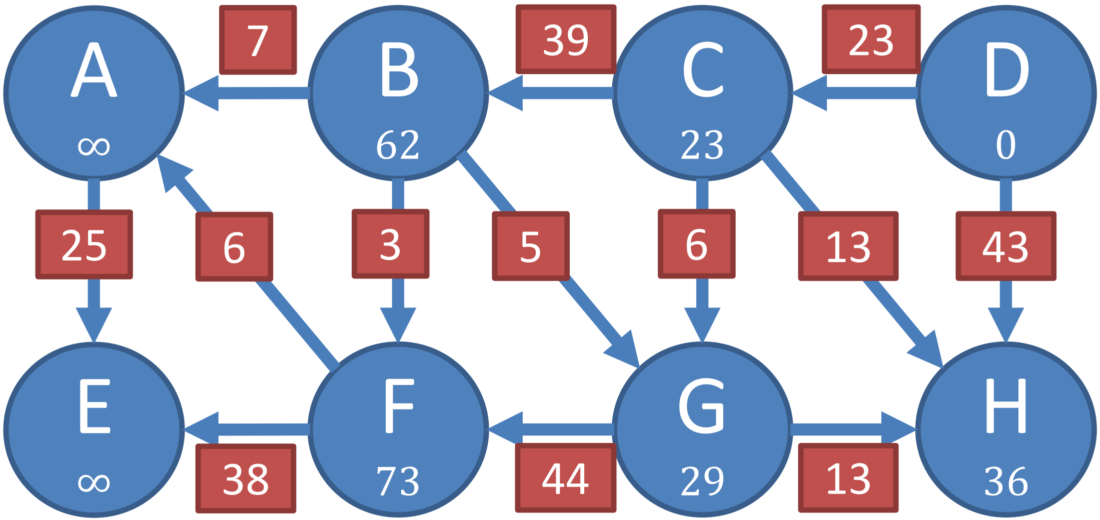
Round 2



Relax order: A B C D E F G H

C->B; C->G; C->H; G->F; G->H

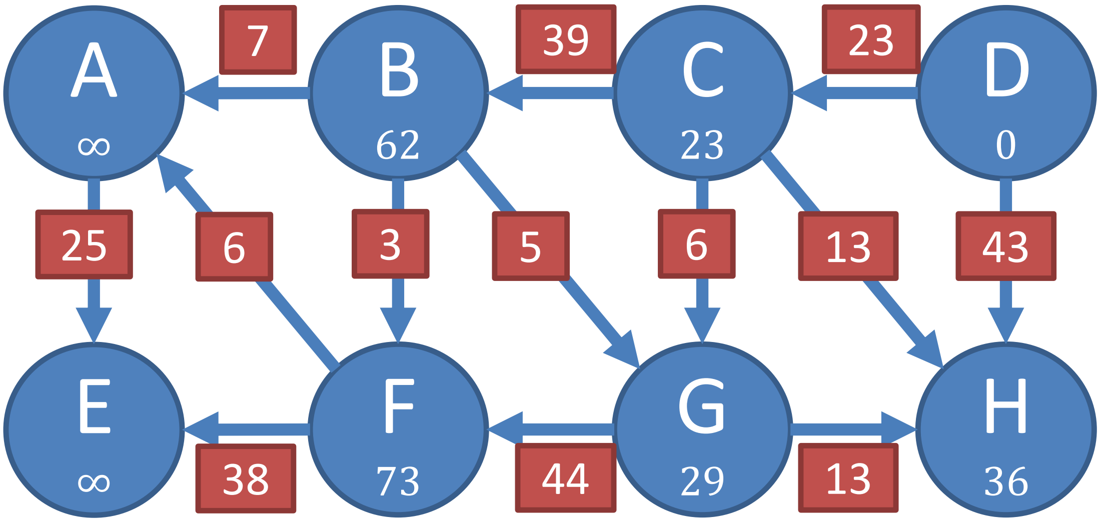
Round 2



Relax order: A B C D E F G H

C->B; C->G; C->H; G->F; G->H

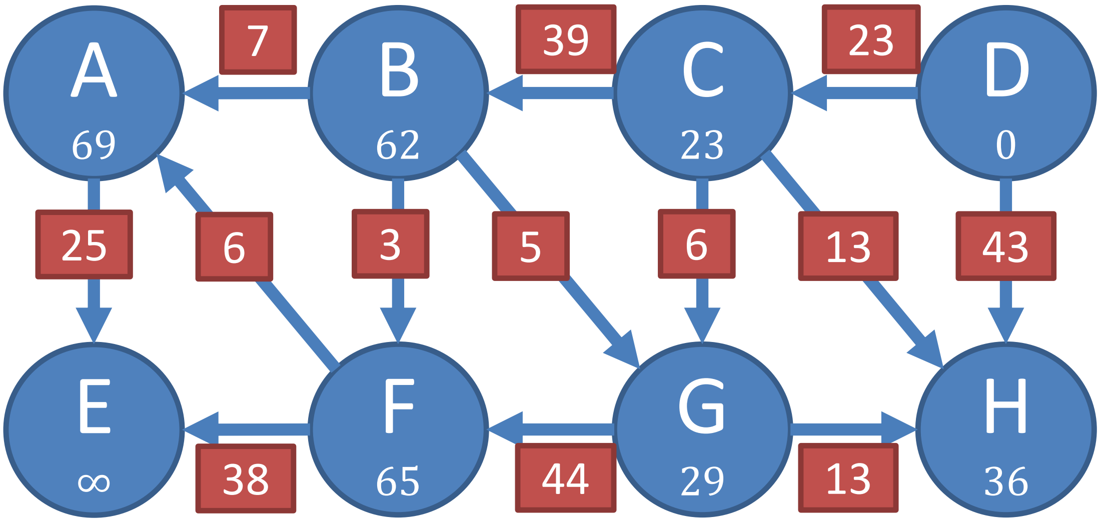
Round 3



Relax order: A B C D E F G H

B->A; B->F; B->G

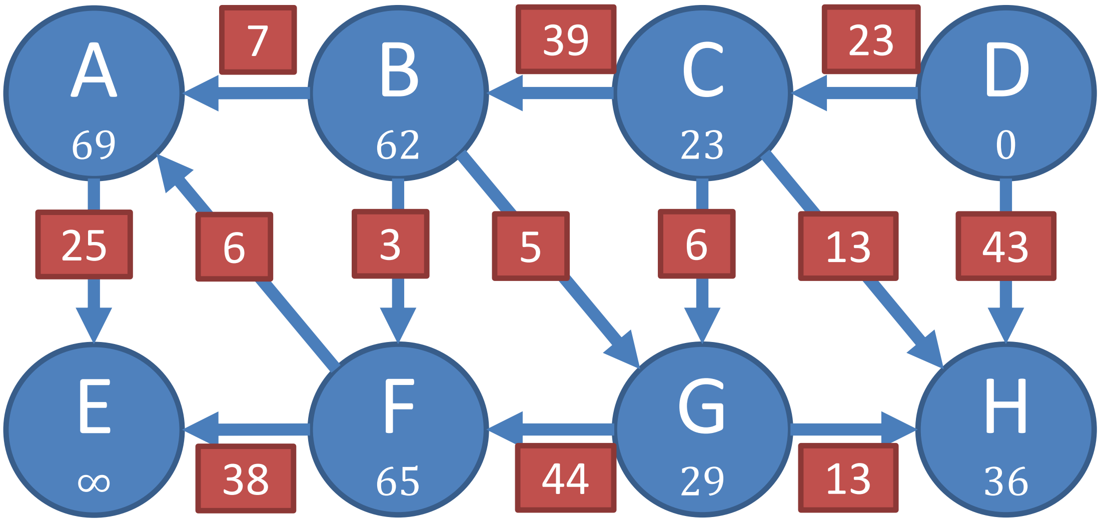
Round 3



Relax order: A B C D E F G H

B->A; B->F; B->G

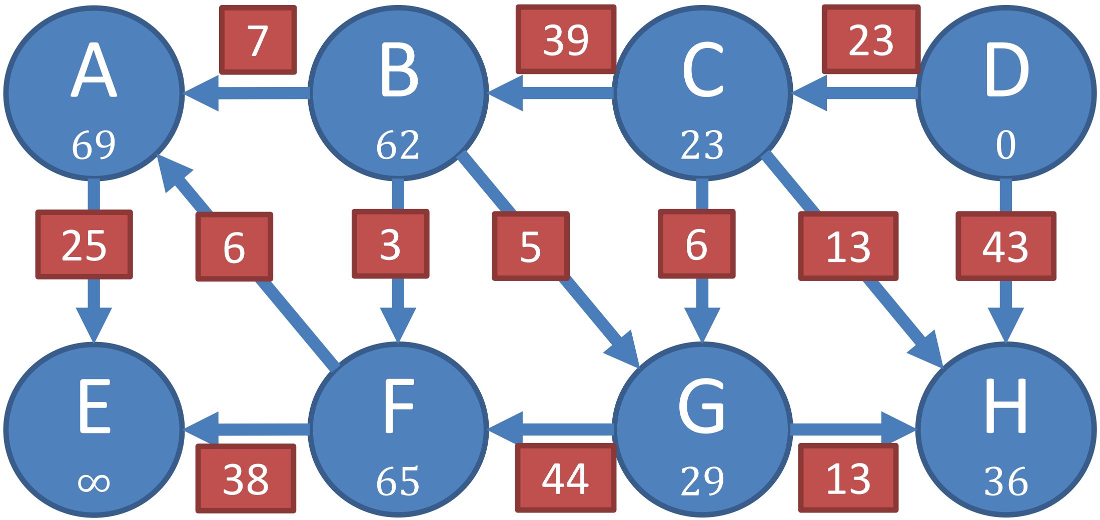
Round 3



Relax order: A B C D E F G H

B->A; B->F; B->G; C, D haven't changed

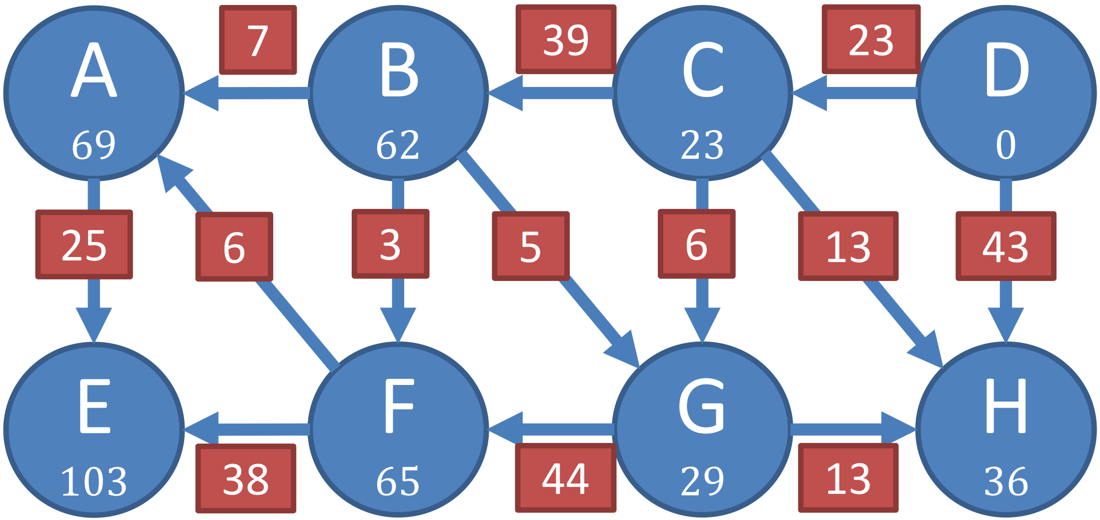
Round 3



Relax order: A B C D E F G H

B->A; B->F; B->G; F->E; F->A

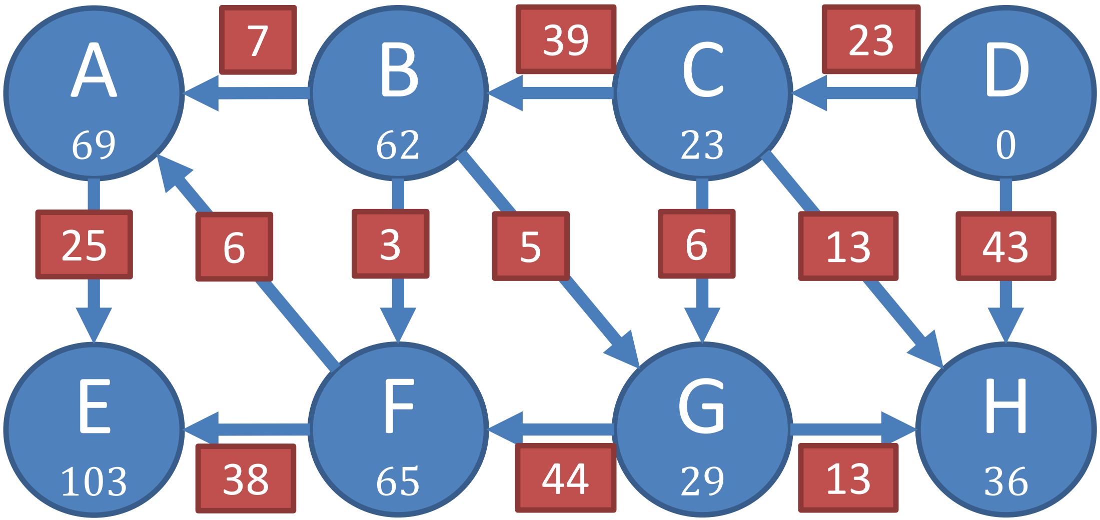
Round 3



Relax order: A B C D E F G H

B->A; B->F; B->G; F->E; F->A

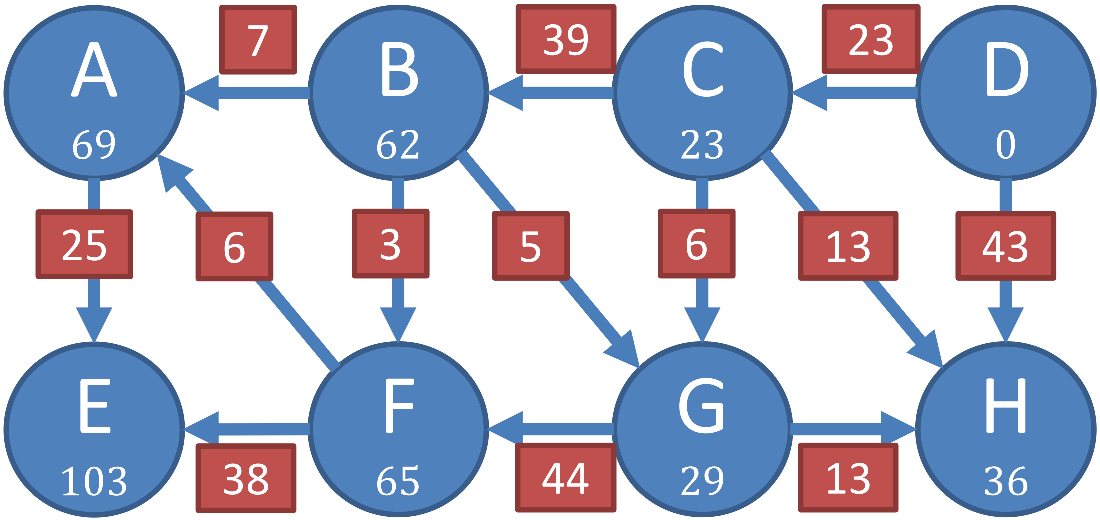
Round 3



Relax order: A B C D E F G H

B->A; B->F; B->G; F->E; F->A; G->F; G->H

Round 3



Relax order: A B C D E F G H

B->A; B->F; B->G; F->E; F->A; G->F; G->H

Back to graph problems

- Calculates $D(s,v)$: the distance from a given source s to all vertices v.
- Solution idea: proceed in rounds; in each round "relax" edges from each vertex in sequence. Assume no negative cycles.
- What is produced after k such relaxations?
- $BF(s,v,k) \leq$ the length of the shortest path from s to v with at most k hops [why not $=$?]
- $D(s,v) = BF(s,v,V)$

Bellman-Ford as DP

- Subproblem: paths with at most k hops.
- Relationship:

$$D(s, v, 0) = \begin{cases} 0 & if\ s = v \\ \infty & oterwise \end{cases}$$

$$D(s, v, k+1) = \min_{u \to v}(D(s, u, k) + c(u \to v))$$

- Note: standard implementation does not compute $D(s, v, k)$, but something potentially smaller.

All-pairs shortest path

- Given a digraph G with no negative cycles, want to compute a table $D(u, v)$ of *all* shortest distances.
- $D(u, v, k)$ =?
- The hard part is coming up with the "right" subproblems!
- Number of steps: possible, but less efficient.

Floyd–Warshall algorithm

- $D(u, v, k)$ = the shortest path where the only intermediate vertices allowed are vertices $\{1, \ldots, k\}$ (assume the vertices are $\{1, \ldots, n\}$)
- $D(u, v) = D(u, v, n)$
- $D(u, v, 0) = c(u \to v)$
- $D(u, v, k + 1)$

$= \min(D(u, v, k), D(u, k + 1, k) + D(k + 1, v, k))$

- Running time?
- $O(V^3)$ (compared to $O(VE)$ for BF).

Subproblem by # of hops

- $L(u, v, k)$ = the shortest path with at most k hops
- $L(u, v) = L(u, v, n)$ (in fact, $L(u, v, n-1)$)
- $L(u, v, 1) = c(u \to v)$
- $L(u, v, k+1)$
$$= \min_{\mathrm{w}}(D(u, w, k) + c(w \to v))$$
- Running time?
- $O(V\ E)$ per update, V updates, $(V^2 E)$ total. (compared to $O(VE)$ for BF: same as running it V times).

Better update?

- $L(u, v, k)$ = the shortest path with at most k hops
- $L(u, v) = L(u, v, n)$
- $L(u, v, 1) = c(u \to v)$
- $L(u, v, 2k) = \min_{\mathrm{w}}(D(u, w, k) + D(w, v, k))$
- Cost per update: $O(V^3)$
- Number of updates: $O(\log V)$
- Total cost $O(V^3 \log V)$ (compared to $O\left(V^3\right)$ for Floyd–Warshall).

Dynamic programming: summary

- Find good subproblems
- Come up with a recursive solution (recurrence relation)
- Use memorization to avoid running time blowup.
- Most important: the right subproblems
- Secondary: better update and storage strategies