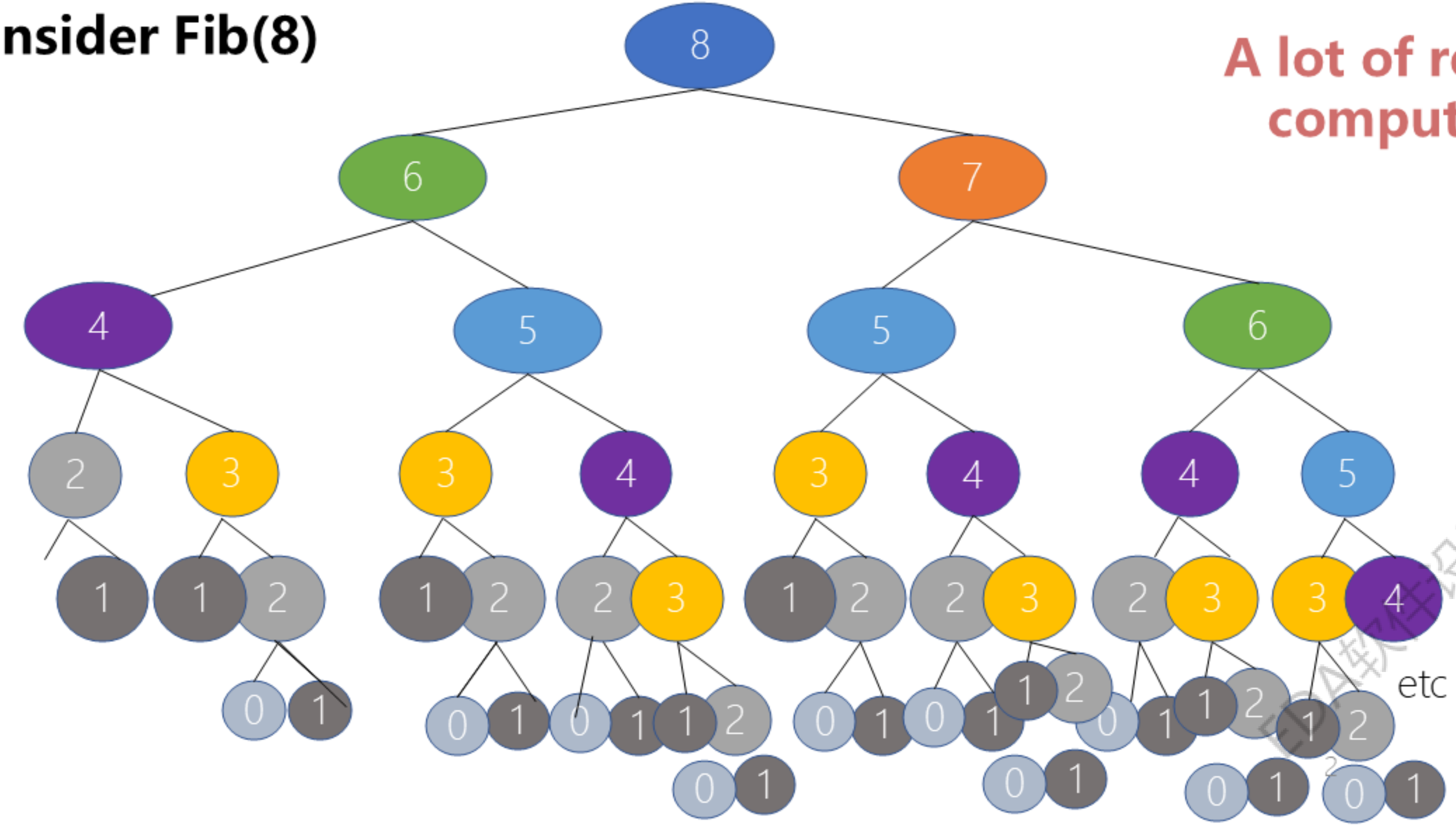
EDA 软件设计 I

Lecture 23

暴力递归的Fibonacci数列

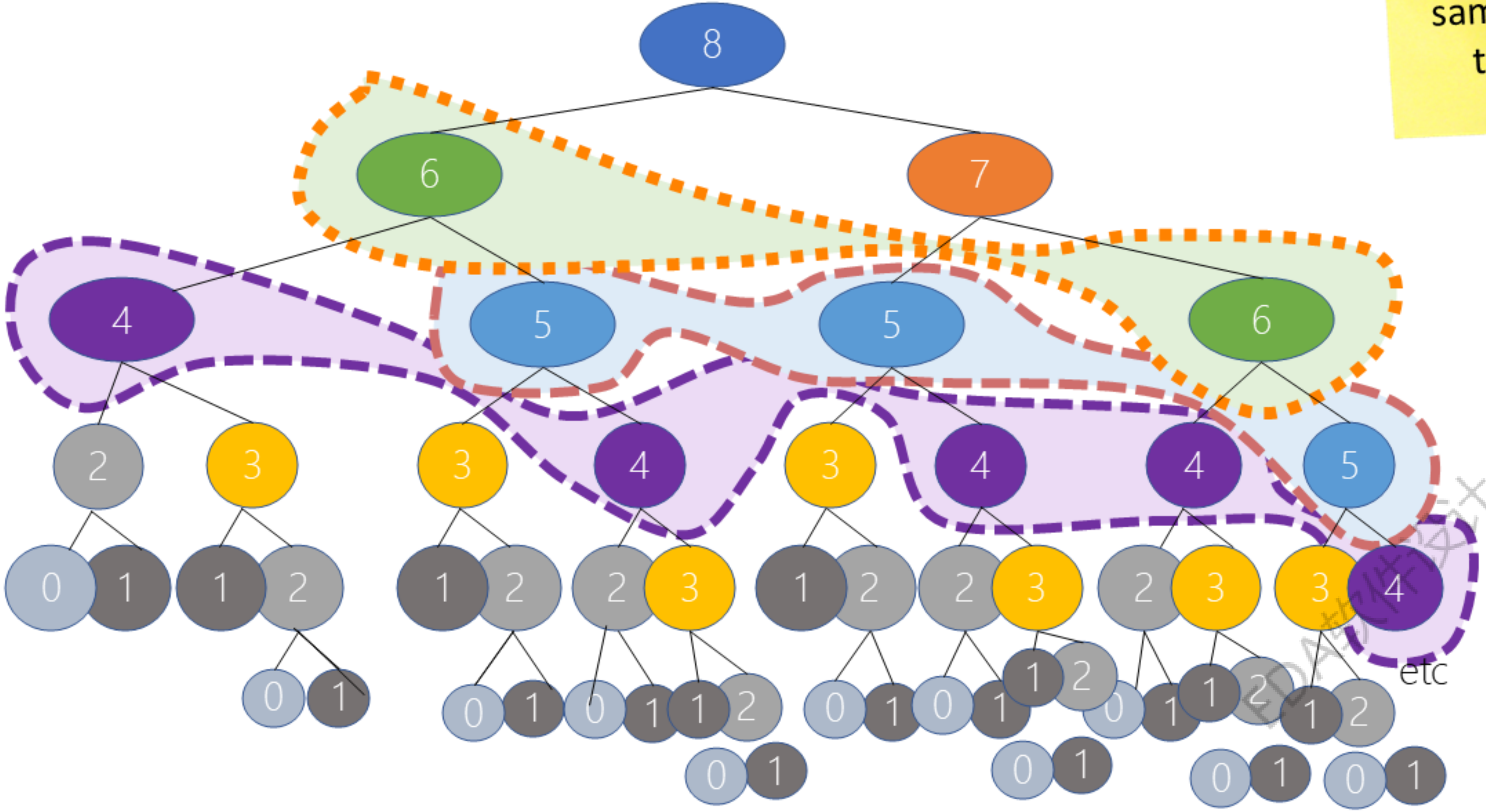
Consider Fib(8)

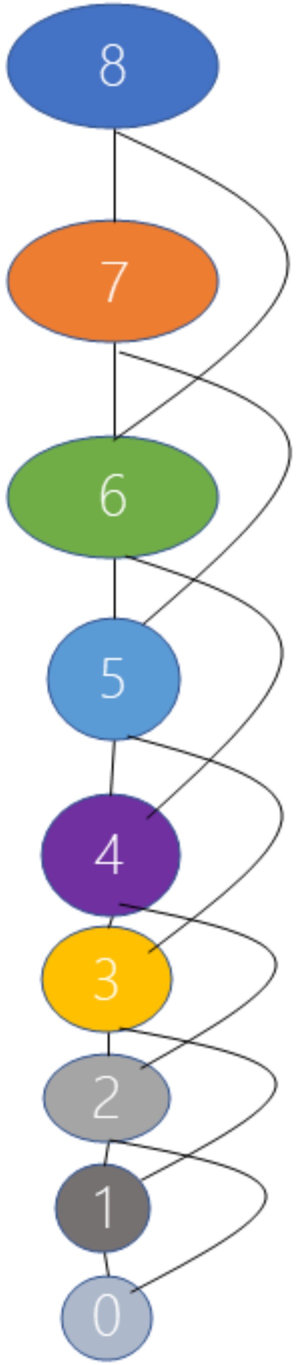


A lot of repeated computation!

Memo-ization visualization

don't do the same work twice!

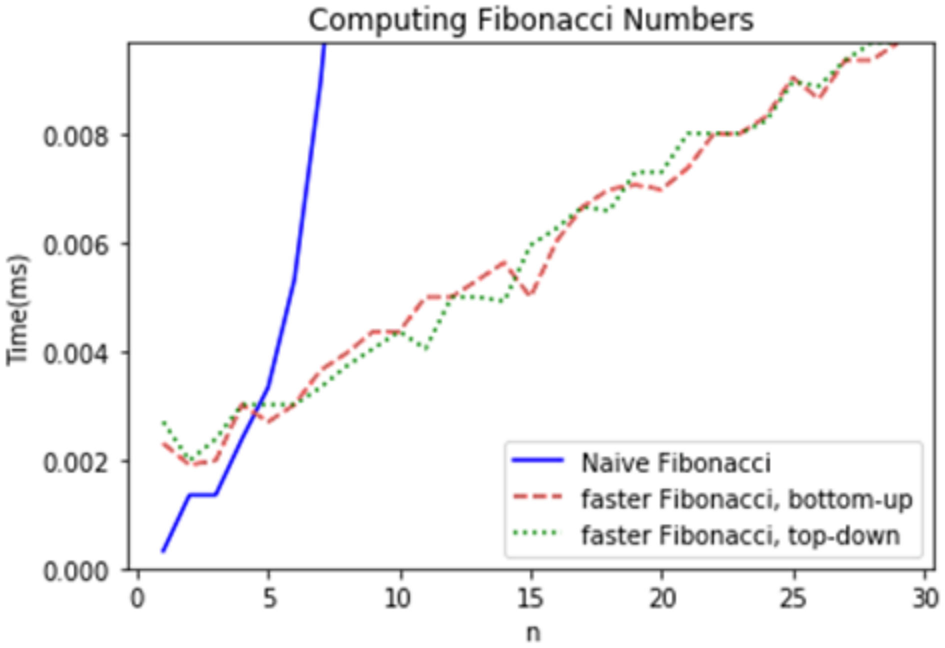




改进后

```
• define a global list F = [0,1,None, None, …, None]
• def Fibonacci(n):
    • if F[n] != None:
        • return F[n]
    • else:
        • F[n] = Fibonacci(n-1) + Fibonacci(n-2)
    • return F[n]
```

Much better running time!



Dynamic Programming!

What is *dynamic programming*?

- **是一种「算法思想」 or「设计范式」**
  - 比如：分治法、贪心、回溯法、分支限界、随机算法
- **Usually, 它是用来解优化问题的**
  - E.g., *shortest* path
  - (Fibonacci numbers aren’t an optimization problem, but they are a good example of DP anyway...)

Why “*dynamic programming*” ？

- **It’s a name coined by Richard Bellman**
- **Programming 指的是“方案”或者“计划”，不是计算机程序**
  - a shortest route is a *plan* aka a *program*.
- **Dynamic 指的是问题具有多阶段的特点**
- **But also it’s just a fancy-sounding name**
  - Bellman当时研究多阶段决策过程（multi-stage decision process），涉及军事背景，选这个名字是为了故意模糊项目性质
  - 避免引起美国国防部高层的反感
  - 选了一个有点神秘的名字

Elements of DP

基本要素/核心组成部分

Elements of DP

1. Optimal sub-structure （最优子结构）：

- 问题能分解成较小的子问题
  - Fibonacci: $F(i)$ for $i \leq n$
- 问题的解可以由更小的子问题的解来推导
  - Fibonacci:

$$F(i+1) = F(i) + F(i-1)$$

Elements of DP

2. Overlapping sub-problems （重叠子问题）：

- 子问题重叠

  Fibonacci:

  - Both F[i+1] and F[i+2] **directly use** F[i].
  - And lots of different F[i+x] **indirectly use** F[i].

- This means that we can **save time** by solving a sub-problem just once and storing the answer.

Design a DP Algorithm

- **Optimal sub-structure**
  - Optimal solutions to sub-problems can be used to find the optimal solution of the original problem.
- **Overlapping sub-problems**
  - The sub-problems show up again and again

- **Using these properties, we can design a *dynamic programming* algorithm:**

  ① Keep a table of solutions to the smaller problems.

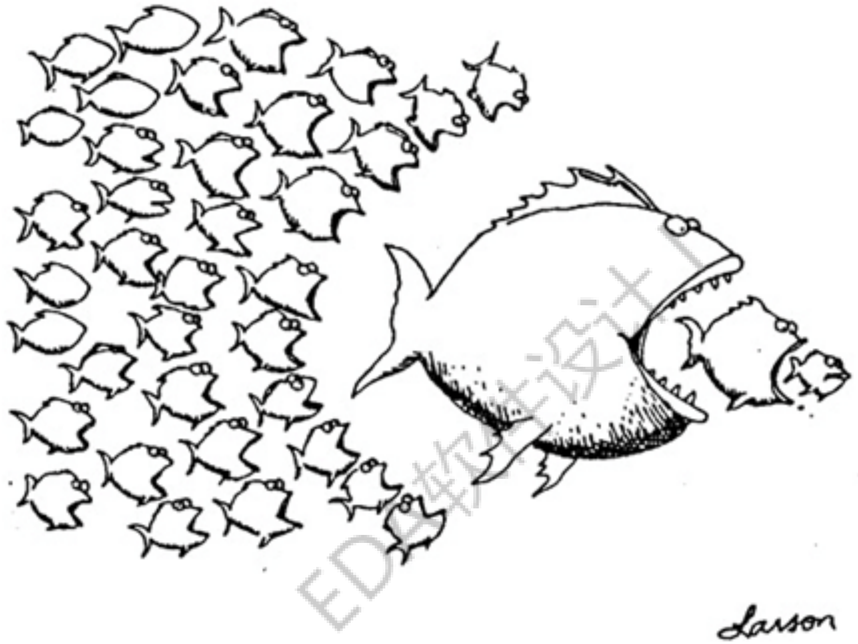
  ② Use the solutions in the table to solve bigger problems.

  ③ At the end we can use information we collected along the way to find the solution to the whole thing.

DP的两种实现方法

- **Top down （自顶向下）**

- **Bottom up（自底向上）**



Top down approach

- 「"记忆化"递归」 or 「 带"备忘录"的递归」
  - 解决大问题
    - 递归地解决小的子问题
      - 再递归地解决更小的子子问题
        - etc..

- 与「分治法」的区别
  - 分治法将问题分为**互不重叠**的子问题并递归求解
  - 而DP将问题分为**可能重叠的子问题**，通过存储子问题的解来避免重复计算—— "memo-ization"

Bottom up approach

- For Fibonacci:
- Solve the small problems first
  - fill in F[0],F[1]
- Then bigger problems
  - fill in F[2]
- …
- Then bigger problems
  - fill in F[n-1]
- Then finally solve the **real problem**
  - fill in F[n]

```
def fasterFibonacci(n):
    • F = [0, 1, None, None, …, None ]
        • \\ F has length n + 1
    • for i = 2, …, n:
        • F[i] = F[i-1] + F[i-2]
    • return F[n]
```

What have we learned?

- ***About Dynamic programming:***
  - 是一种算法的「设计范式/原理」
  - 利用 **optimal substructure**
  - 利用 **overlapping subproblems**
  - 实现方式： **bottom-up** or **top-down**.
  - It’s a fancy name for a pretty common-sense idea:

Dijkstra is an example of…

Greedy

Bellman-Ford is an example of…

Dynamic Programming

Why B-F is a DP Algorithm?

Bellman-Ford

- **有最优子结构性质?**
- **重叠子问题是?**

Review: Dijkstra vs. Bellman-Ford

- **Dijkstra idea:**
  - Find the u with the smallest d[u]
  - Update u's neighbors: d[v] = min(d[v], d[u] + w(u,v))

- **Bellman-Ford idea:**
  - Don't bother finding the u with the smallest d[u]
  - Everyone updates!

Review: Dijkstra vs. Bellman-Ford

- **Dijkstra:**
  - Needs non-negative edge weights
  - If the weights change, we need to re-run the whole thing.
- **Bellman-Ford:**
  - (-) Slower than Dijkstra's algorithm
    - 二者的时间复杂度分别是什么？不同的实现方式对应不同的时间复杂度
  - (+) Can handle negative edge weights
  - (+) Allows for some flexibility if the weights change

Review: Bellman-Ford Algorithm

G = (V, E) is a graph with n vertices and m edges.

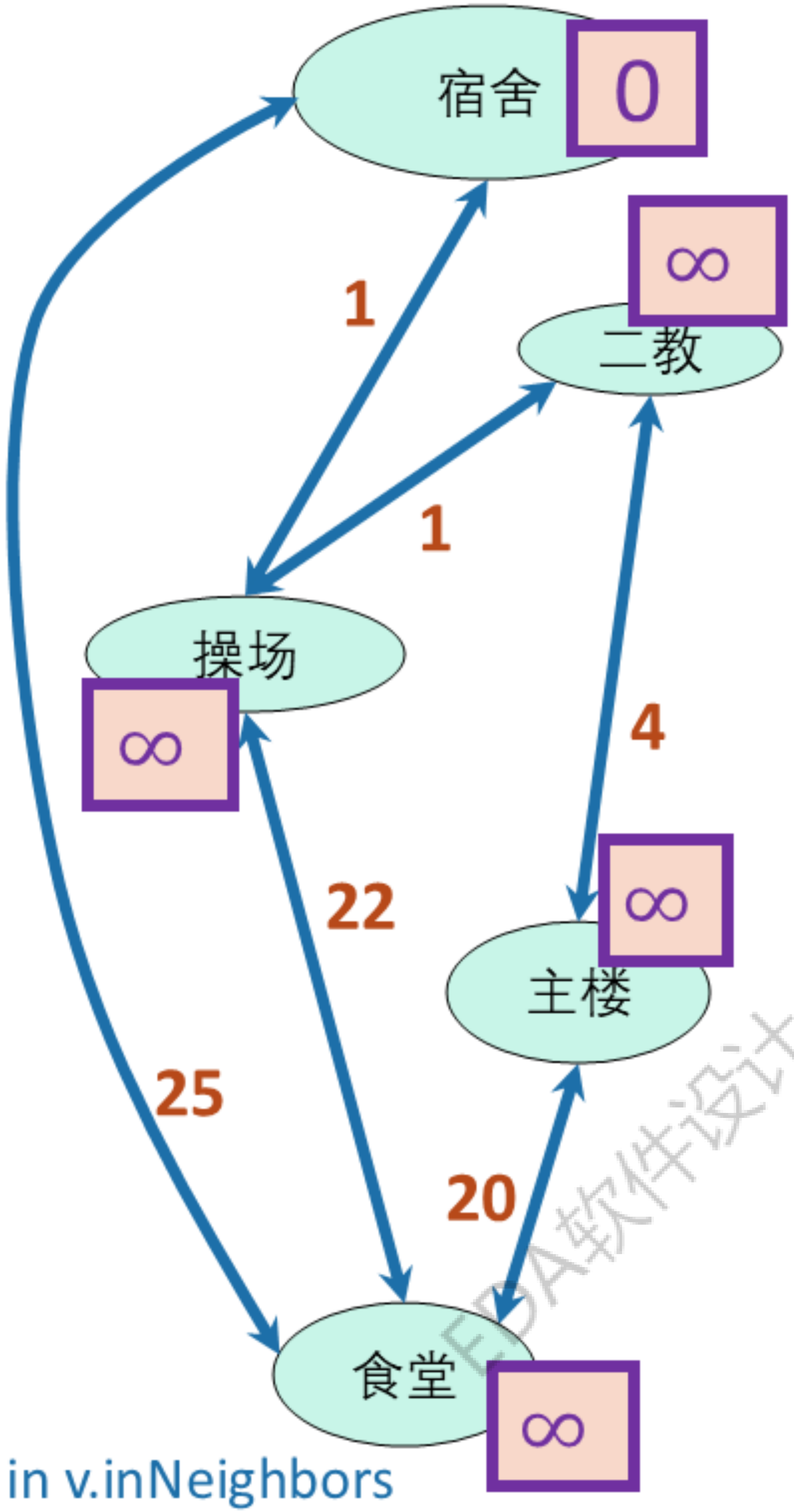
- $d^{(0)}[v] = \boldsymbol{U}$ for all v, where $\boldsymbol{U}$ is a very large number
- $d^{(0)}[s] = 0$
- For i=0,…,n-1:
  - For v in V:
    - $d^{(i+1)}[v] \leftarrow \min(\, d^{(i)}[v] \,,\, \min_{u \text{ in } v.\text{inNeighbors}} \{d^{(i)}[u] + w(u,v)\}\,)$
- If $d^{(n-1)} \neq d^{(n)}$:
  - Return NEGATIVE CYCLE ☹
- Otherwise, $\text{dist}(s,v) = d^{(n-1)}[v]$

Here, Dijkstra picked a special vertex u and updated u's neighbors – Bellman-Ford will update all the vertices.

Bellman-Ford

How far is a node from 宿舍?

	宿舍	操场	二教	主楼	食堂
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$					
$d^{(2)}$					
$d^{(3)}$					
$d^{(4)}$					



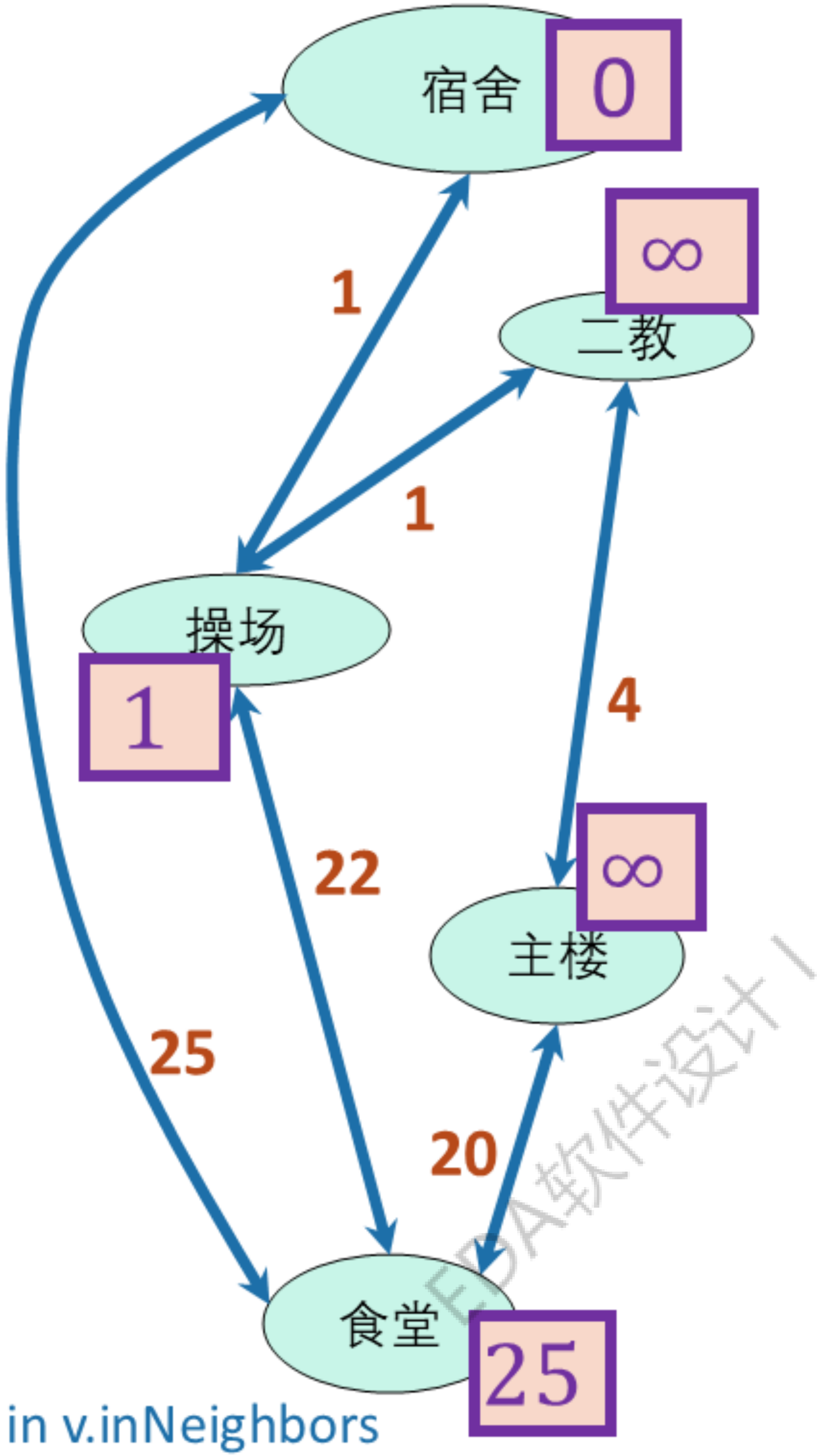
- **For** i=0,…,n-2:
  - **For** v in V:
    - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , d^{(i)}[u] + w(u,v))$
      where we are also taking the min over all u in v.inNeighbors

Bellman-Ford

How far is a node from 宿舍?

	宿舍	操场	二教	主楼	食堂
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$					
$d^{(3)}$					
$d^{(4)}$					

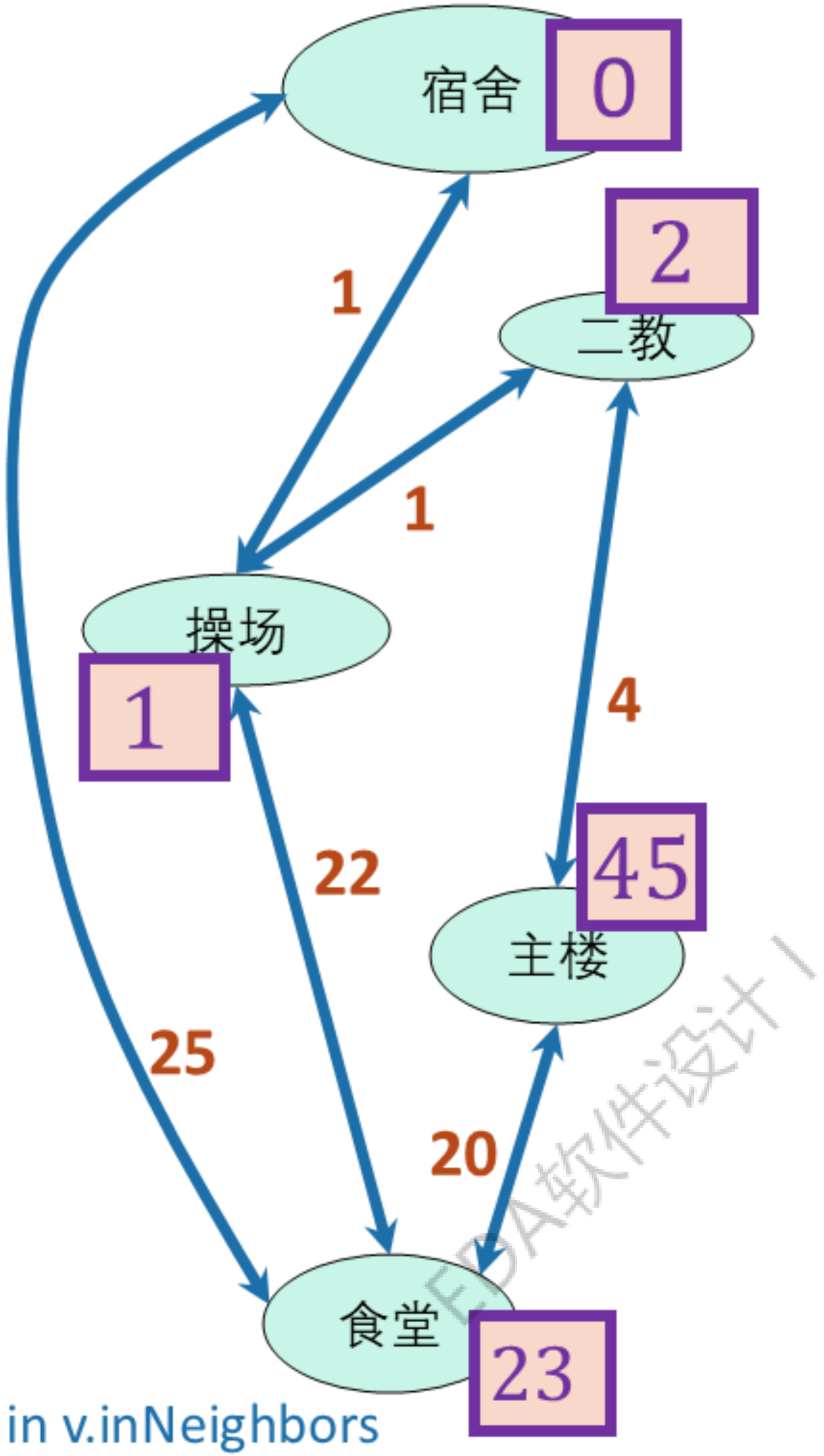
- **For** i=0,…,n-2:
  - **For** v in V:
    - $d^{(i+1)}[v] \leftarrow \min(\, d^{(i)}[v]\, ,\, d^{(i)}[u] + w(u,v)\,)$
      where we are also taking the min over all u in v.inNeighbors



Bellman-Ford

How far is a node from 宿舍?

	宿舍	操场	二教	主楼	食堂
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$	0	1	2	45	23
$d^{(3)}$					
$d^{(4)}$					



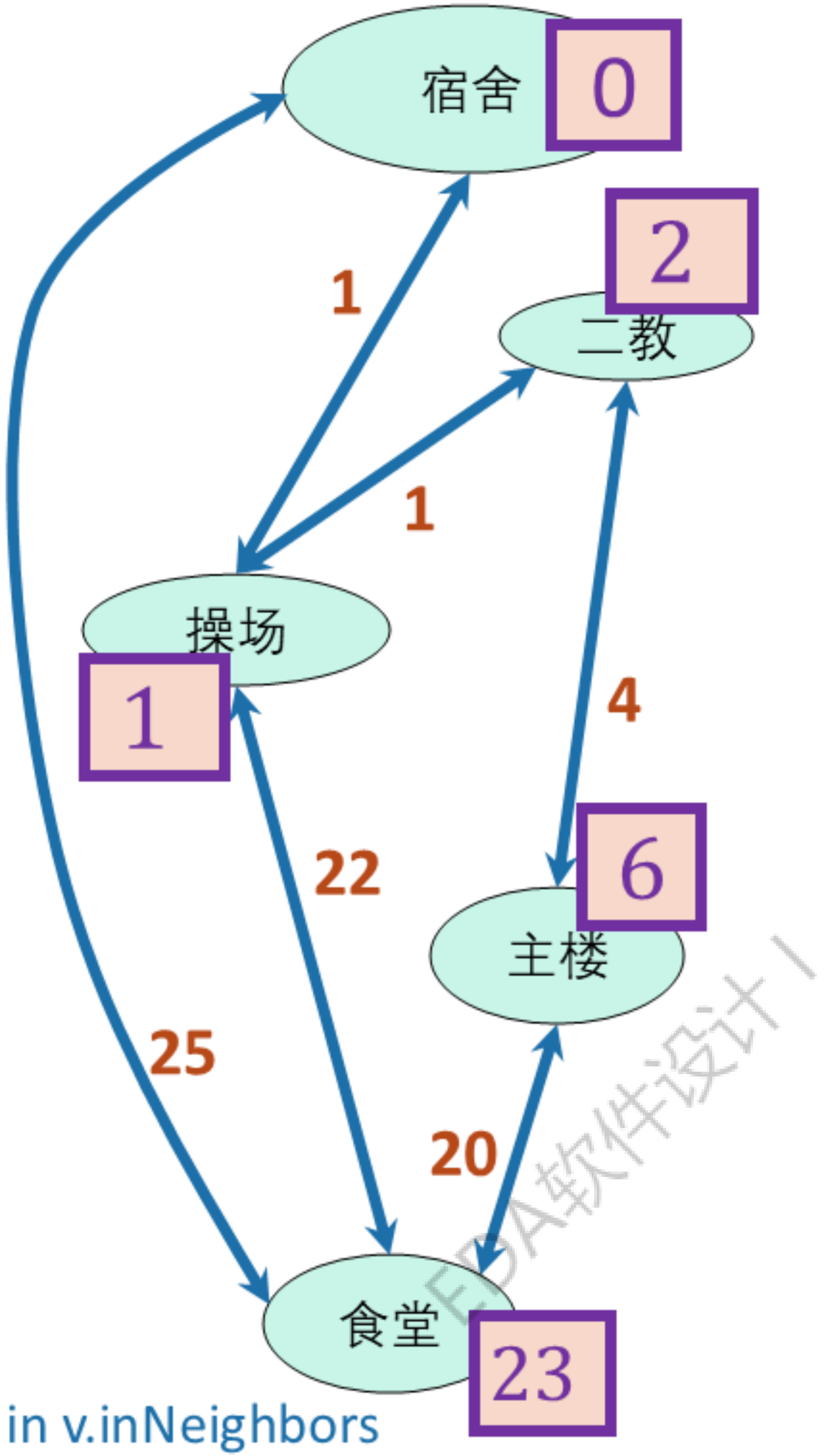
- **For** i=0,…,n-2:
  - **For** v in V:
    - $d^{(i+1)}[v] \leftarrow \min(\, d^{(i)}[v]\,,\, d^{(i)}[u] + w(u,v)\,)$
      where we are also taking the min over all u in v.inNeighbors

Bellman-Ford

How far is a node from 宿舍?

	宿舍	操场	二教	主楼	食堂
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$	0	1	2	45	23
$d^{(3)}$	0	1	2	6	23
$d^{(4)}$					

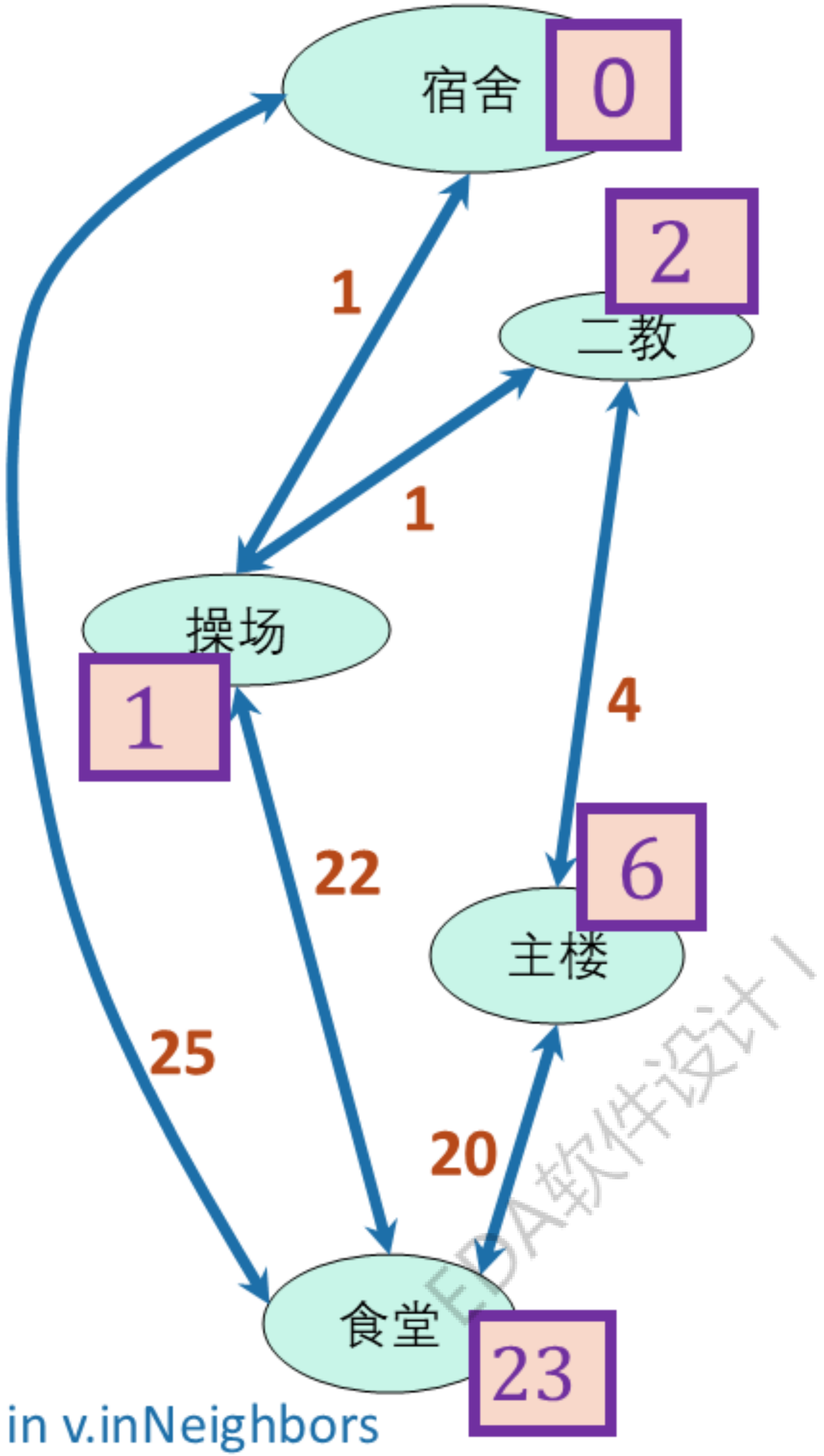
- **For** i=0,…,n-2:
  - **For** v in V:
    - $d^{(i+1)}[v] \leftarrow \min(\, d^{(i)}[v]\ ,\ d^{(i)}[u] + w(u,v)\,)$
      where we are also taking the min over all u in v.inNeighbors



Bellman-Ford

How far is a node from 宿舍?

	宿舍	操场	二教	主楼	食堂
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$	0	1	2	45	23
$d^{(3)}$	0	1	2	6	23
$d^{(4)}$	0	1	2	6	23



- **For** i=0,…,n-2:
  - **For** v in V:
    - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , d^{(i)}[u] + w(u,v))$
      where we are also taking the min over all u in v.inNeighbors

Note on implementation：如何优化空间

- Don’t actually keep all n arrays around.
- Just keep two at a time: “last round” and “this round”

	宿舍	操场	二教	主楼	食堂
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	1	∞	∞	25
$d^{(2)}$	0	1	2	45	23
$d^{(3)}$	0	1	2	6	23
$d^{(4)}$	0	1	2	6	23

Only need these two in order to compute $d^{(4)}$

We don’t even need two, just one array is fine. Why and how?

Bellman-Ford Summary

- Running time is O(mn)
  - For each of n rounds, update m edges.
- Works fine with negative edges.
- Does not work with negative cycles.
  - No algorithm can – shortest paths aren't defined if there are negative cycles.
- B-F can detect negative cycles!
- For your information: by now we have faster (but complicated) algorithms with runtime $\simeq O(m\log(n)^c)$ as long as weights are not too large in magnitude!

[Bernstein-Nanongkai-Wulff-Nilsen'2022 (FOCS)]

Technically, the weights need to be integers, and then the runtime scales linearly with $\log(W)$ where W is the largest absolute value of the weights.

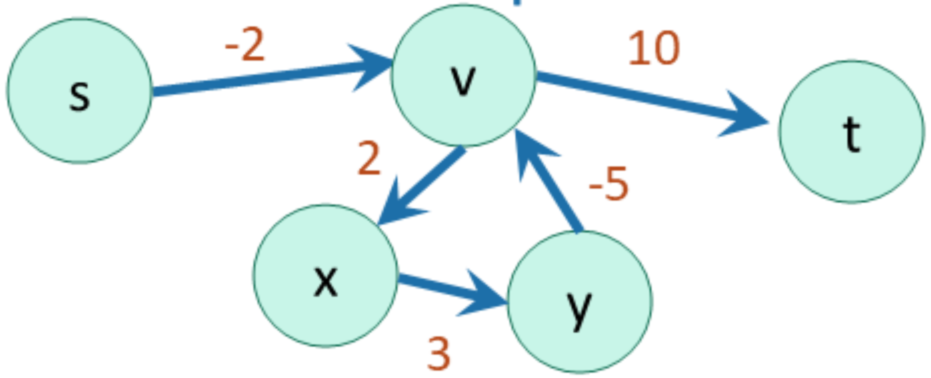
Why does Bellman-Ford work?

- Homework 1: the correctness of Bellman-Ford
- 可用数学归纳法:
  - Inductive hypothesis:
    - $d^{(i)}[v]$ is equal to the cost of the shortest path between s and v **with at most i edges**.
  - Conclusion:
    - $d^{(n-1)}[v]$ is equal to the cost of the shortest path between s and v **with at most n-1 edges**.
    - **If there are no negative cycles**, $d^{(n-1)}[v]$ is equal to the cost of the shortest path.

Notice that negative edge weights are fine.
Just not negative cycles.

Hints of Homework 1:

- Assume there is no negative cycle, then there is a shortest path from s to t, and moreover there is a simple shortest path.



This cycle isn't helping.
Just get rid of it.

- A simple path in a graph with n vertices has at most n-1 edges in it.
- So there is a shortest path with at most n-1 edges

"Simple" means that the path has no cycles in it.