Design and Analysis of Algorithms

Part IV: Graph Algorithms

Lecture 31: All-Pairs Shortest Paths

童咏昕

北京航空航天大学
计算机学院

图算法篇概述

- 在算法课程第四部分“图算法”主题中，我们将主要聚焦于如下经典问题：
  - Basic Concepts in Graph Algorithms（图算法的基本概念）
  - Breadth-First Search (BFS，广度优先搜索）
  - Depth-First Search (DFS，深度优先搜索）
  - Cycle Detection（环路检测）
  - Topological Sort（拓扑排序）
  - Strongly Connected Components（强连通分量）
  - Minimum Spanning Trees (最小生成树)
  - Single Source Shortest Path (单源最短路径)
  - All-Pairs Shortest Paths (所有点对最短路径)
  - Bipartite Graph Matching（二分图匹配）
  - Maximum/Network Flows (最大流/网络流)

提纲

问题定义

算法思想

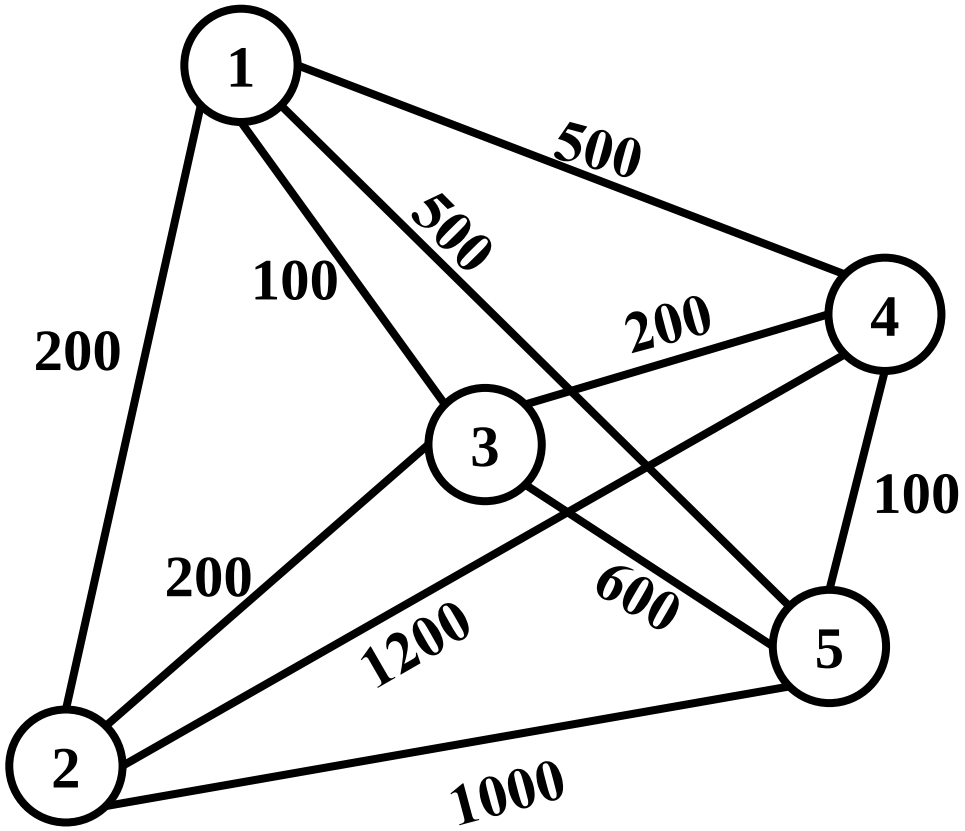
算法设计

算法实例

算法分析

问题背景

- 航班价格



如何求出所有城市之间的最低航班价格?

问题定义

所有点对最短路径问题

All Pairs Shortest Paths

输入

- 带权无向图$G =< V, E, W >$，W为边权

输出

- $\forall u, v \in V$, 从u到v的最短路径

提纲

问题定义

算法思想

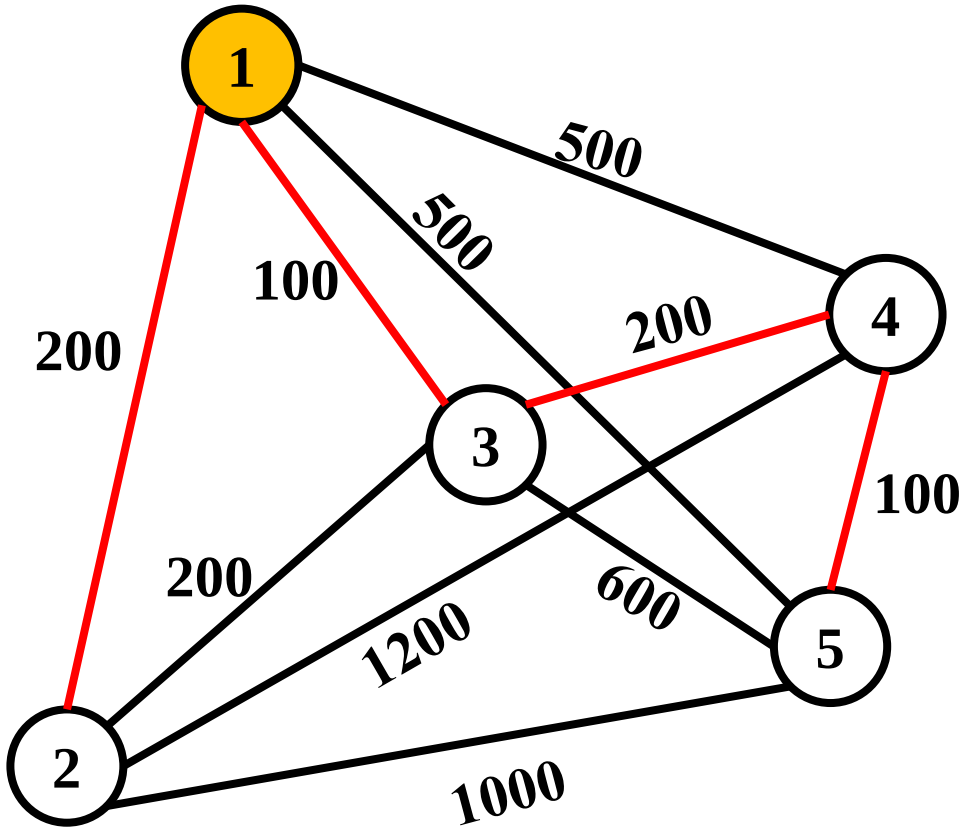
算法设计

算法实例

算法分析

直观思路

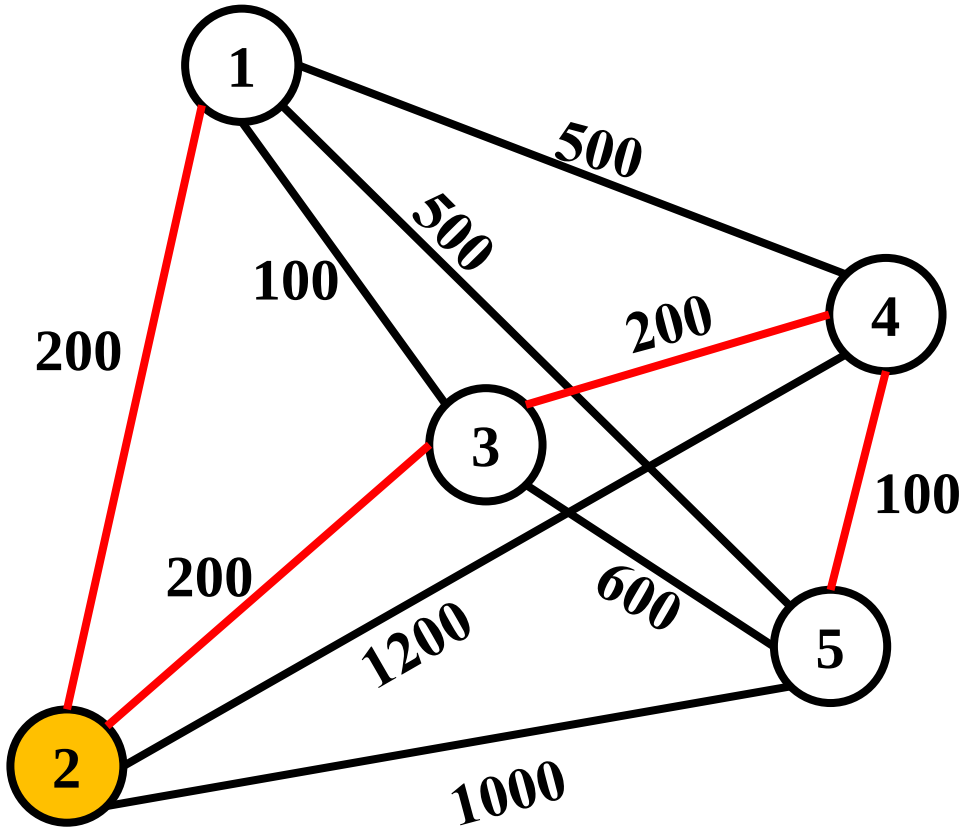
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	200	100	300	400
2					
3					
4					
5					

直观思路

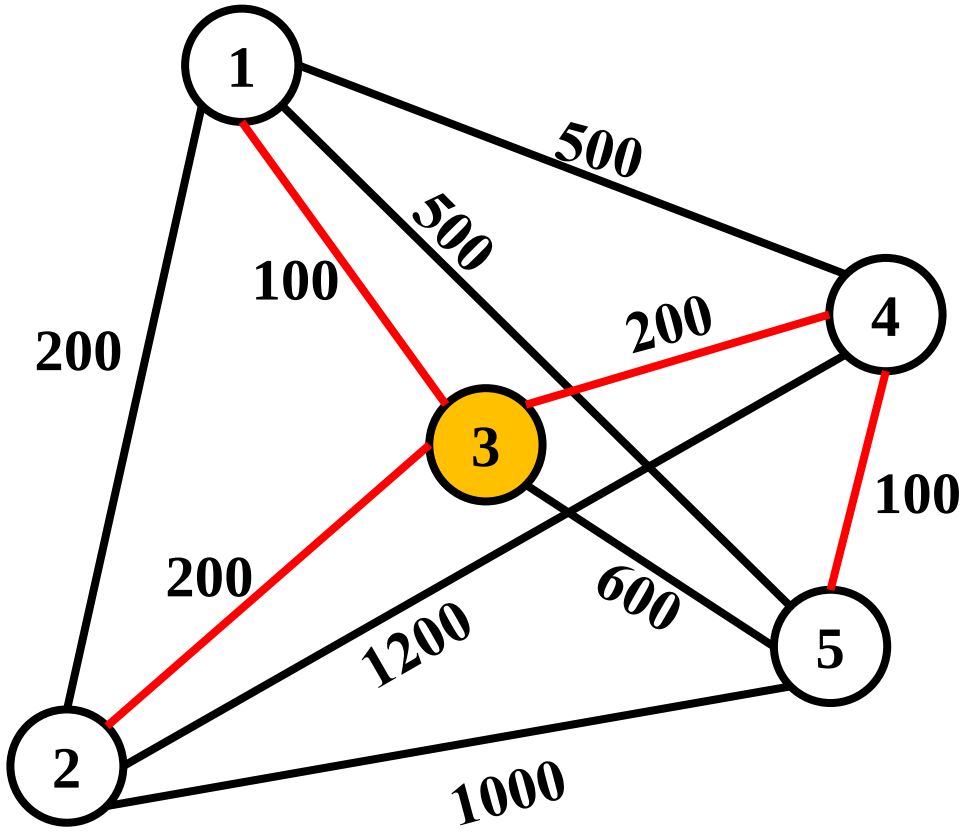
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3					
4					
5					

直观思路

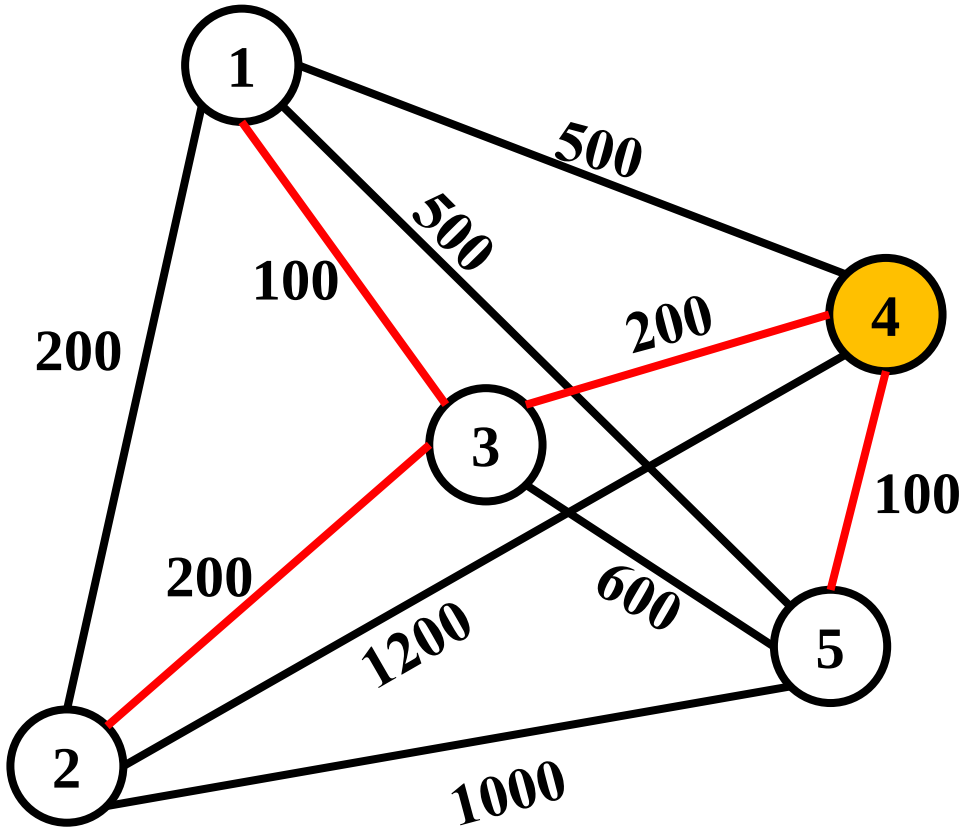
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4					
5					

直观思路

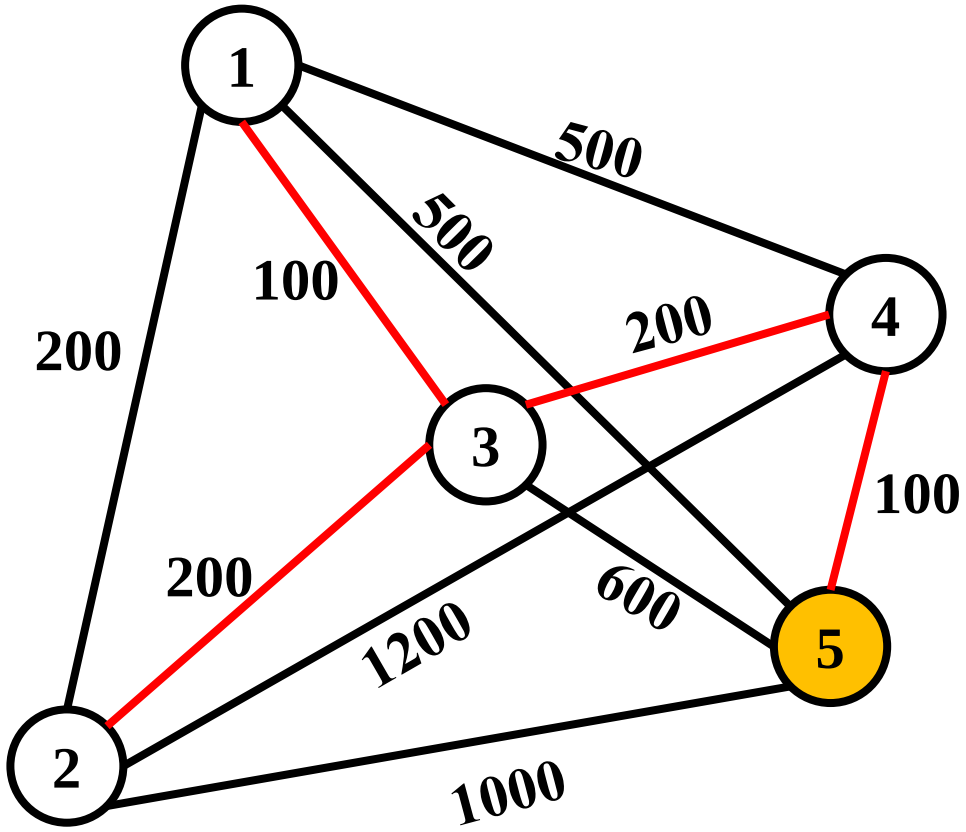
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5					

直观思路

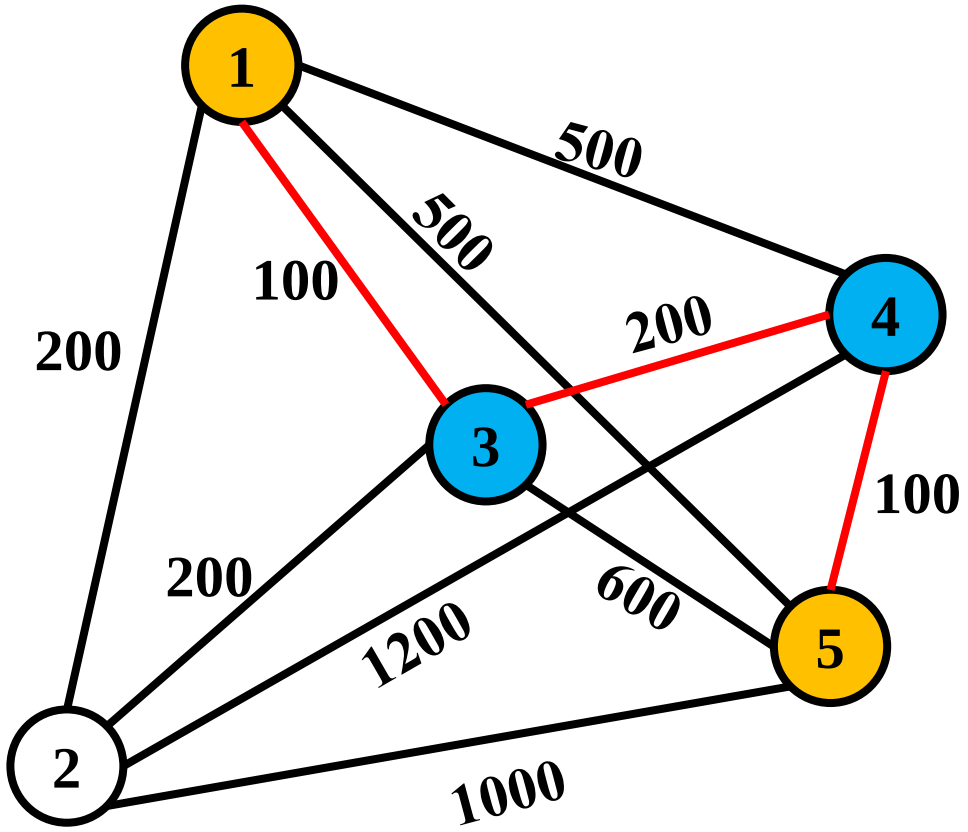
- 使用Dijkstra算法依次求解所有点



v / u	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

直观思路

- 使用Dijkstra算法依次求解所有点
- 存在重叠子问题

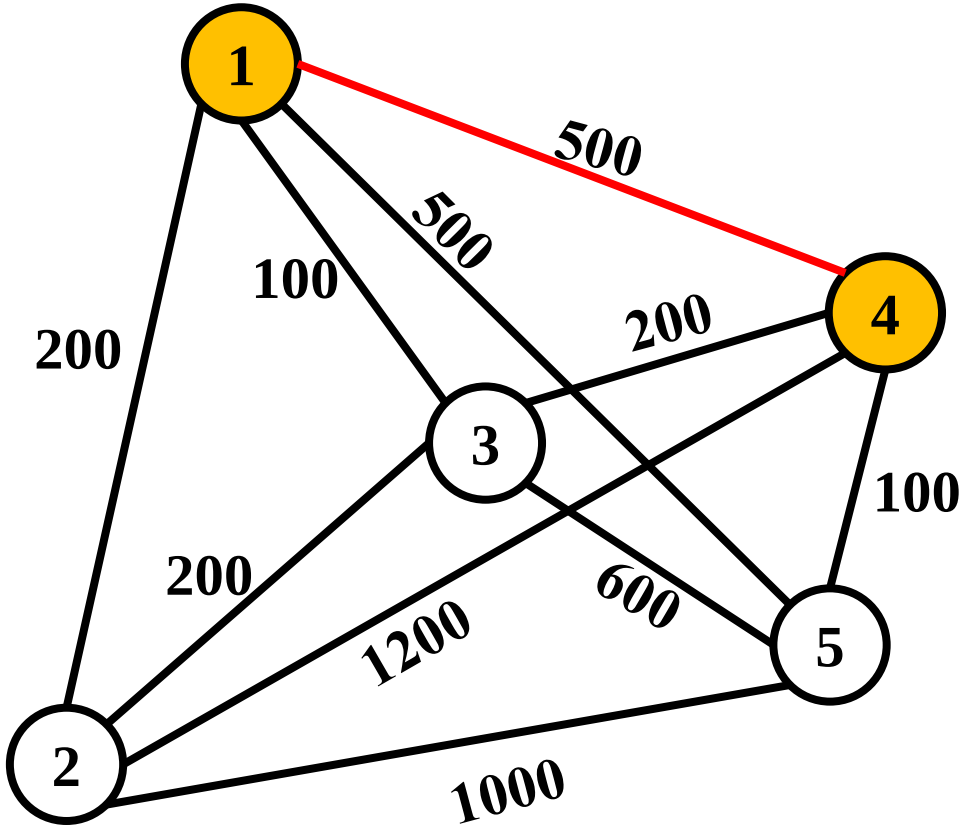


从1到5的最短路径：1→3→4→5

从3到5的最短路径：3→4→5

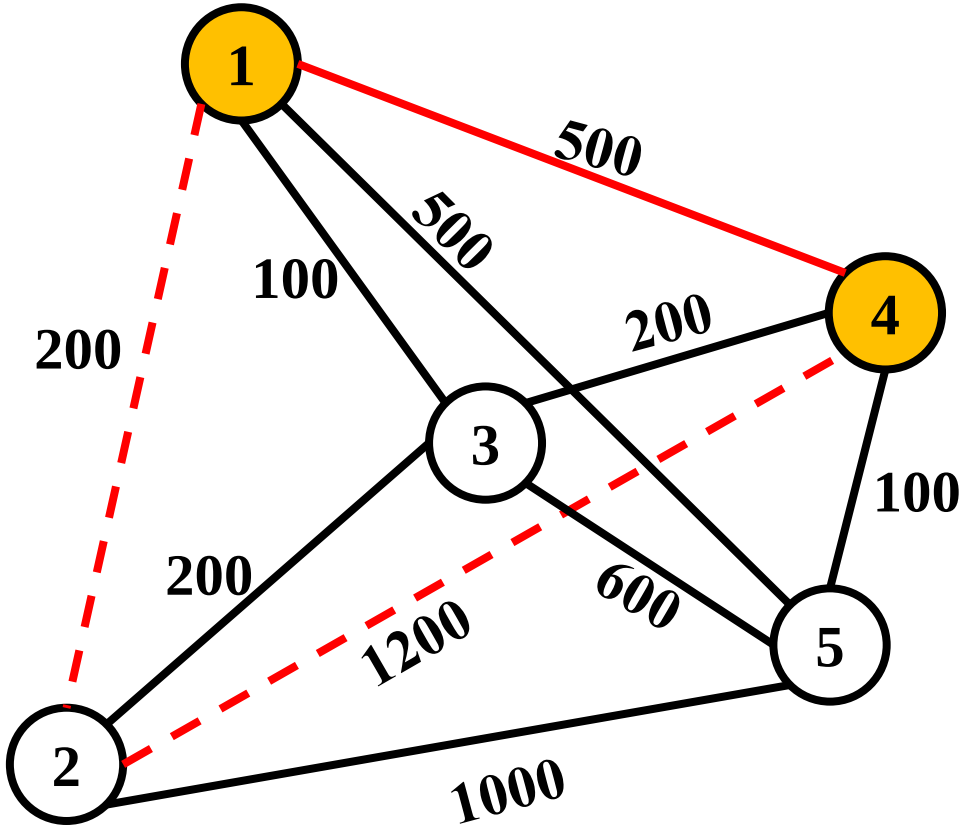
观察松弛过程

- 从1到4的路径更新
  - 可从前1个点中选择点经过：500



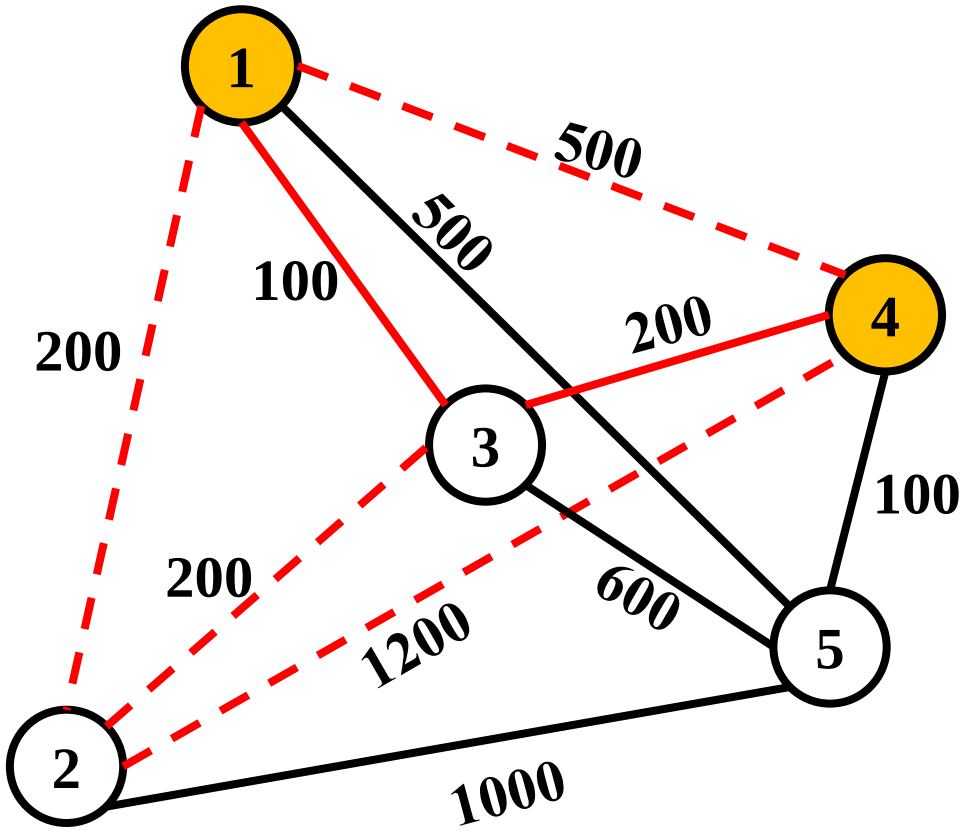
观察松弛过程

- 从1到4的路径更新
  - 可从前1个点中选择点经过：500
  - 可从前2个点中选择点经过：500



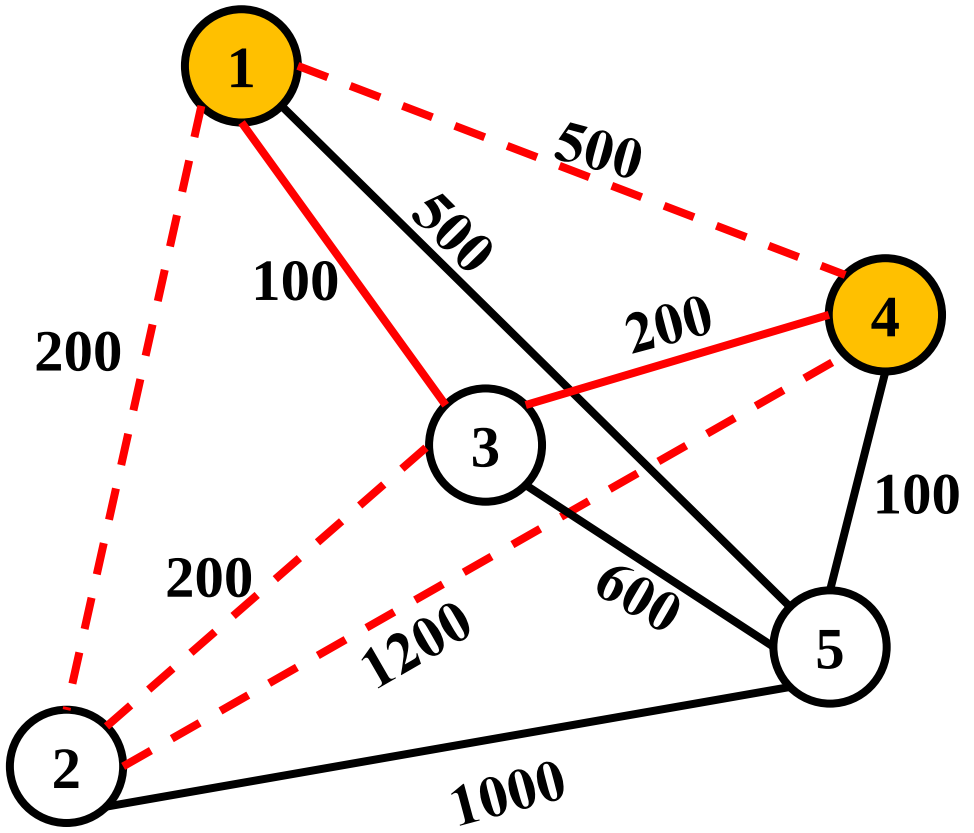
观察松弛过程

- 从1到4的路径更新
  - 可从前1个点中选择点经过：500
  - 可从前2个点中选择点经过：500
  - 可从前3个点中选择点经过：300



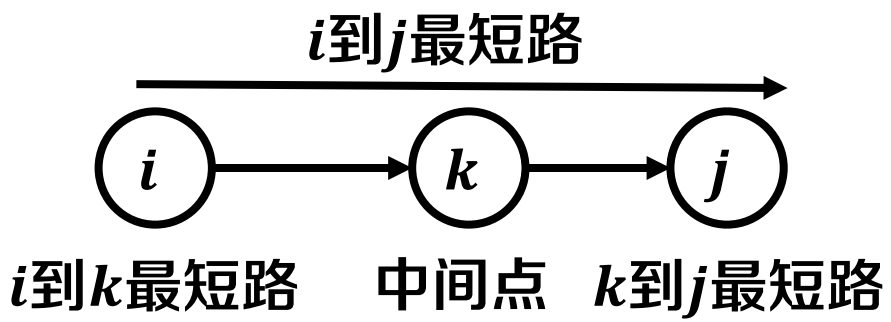
观察松弛过程

- 从1到4的路径更新
  - 可从前1个点中选择点经过：500
  - 可从前2个点中选择点经过：500
  - 可从前3个点中选择点经过：300
  - 可从前k个点中选择点经过：…

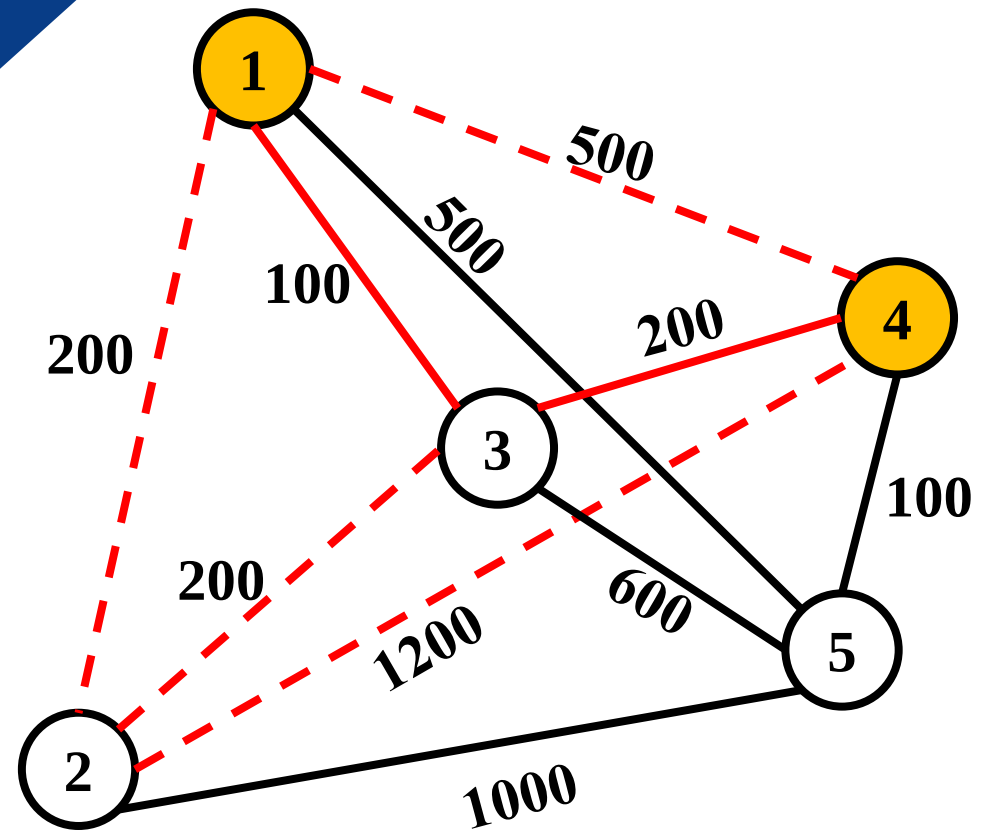


观察松弛过程

- 从1到4的路径更新
  - 可从前1个点中选择点经过：500
  - 可从前2个点中选择点经过：500
  - 可从前3个点中选择点经过：300
  - 可从前k个点中选择点经过：…

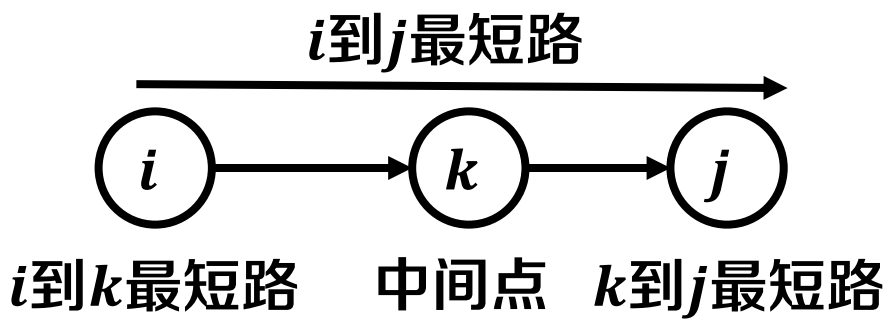


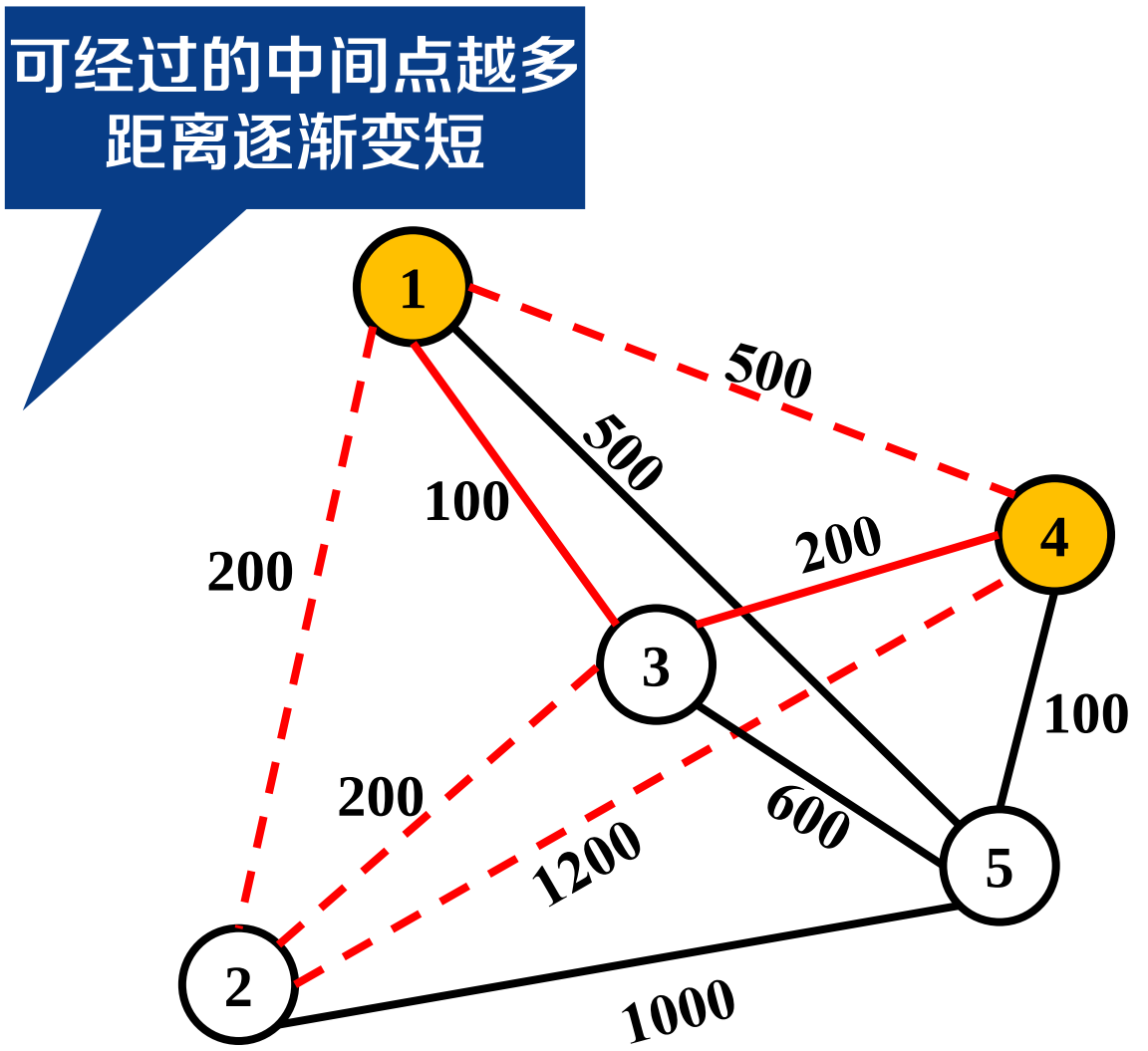
可经过的中间点越多
距离逐渐变短



观察松弛过程

- 从1到4的路径更新
  - 可从前1个点中选择点经过：500
  - 可从前2个点中选择点经过：500
  - 可从前3个点中选择点经过：300
  - 可从前k个点中选择点经过：…





重叠子问题、最优子结构启发使用动态规划求解

提纲

问题定义

算法思想

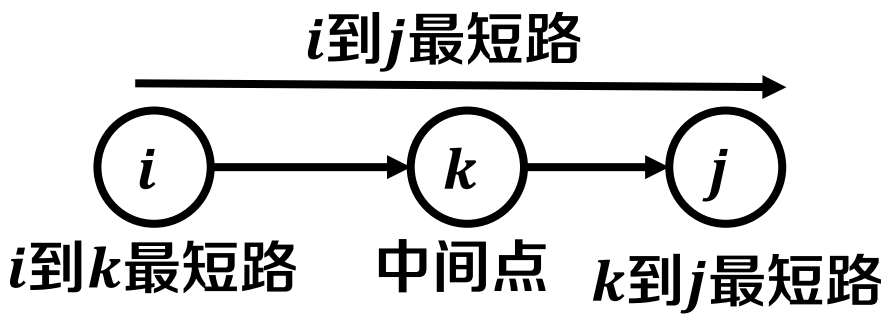
算法设计

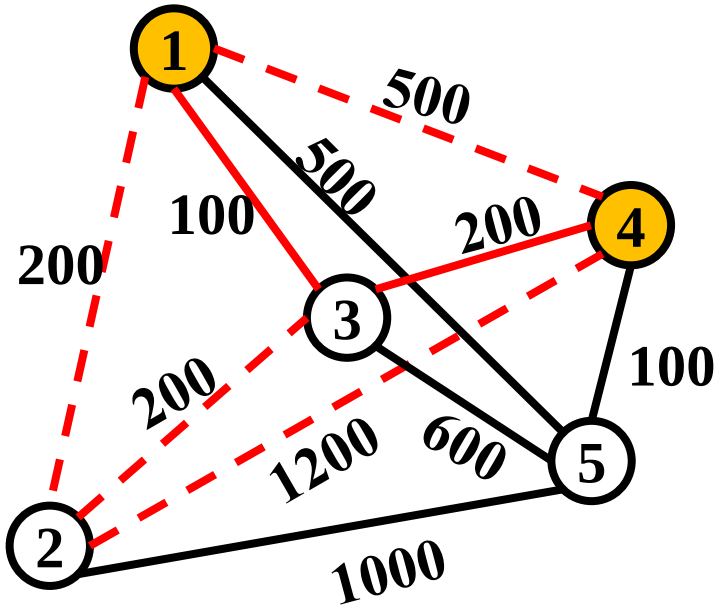
算法实例

算法分析

动态规划：问题结构分析

- 给出问题表示
  - $D[k,i,j]$：可从前k个点选点经过时，i到j的最短距离
- 从1到4的路径更新
  - 可从前1个点中选择点经过：$D[1,1,4]=500$
  - 可从前2个点中选择点经过：$D[2,1,4]=500$
  - 可从前3个点中选择点经过：$D[3,1,4]=300$





问题结构分析

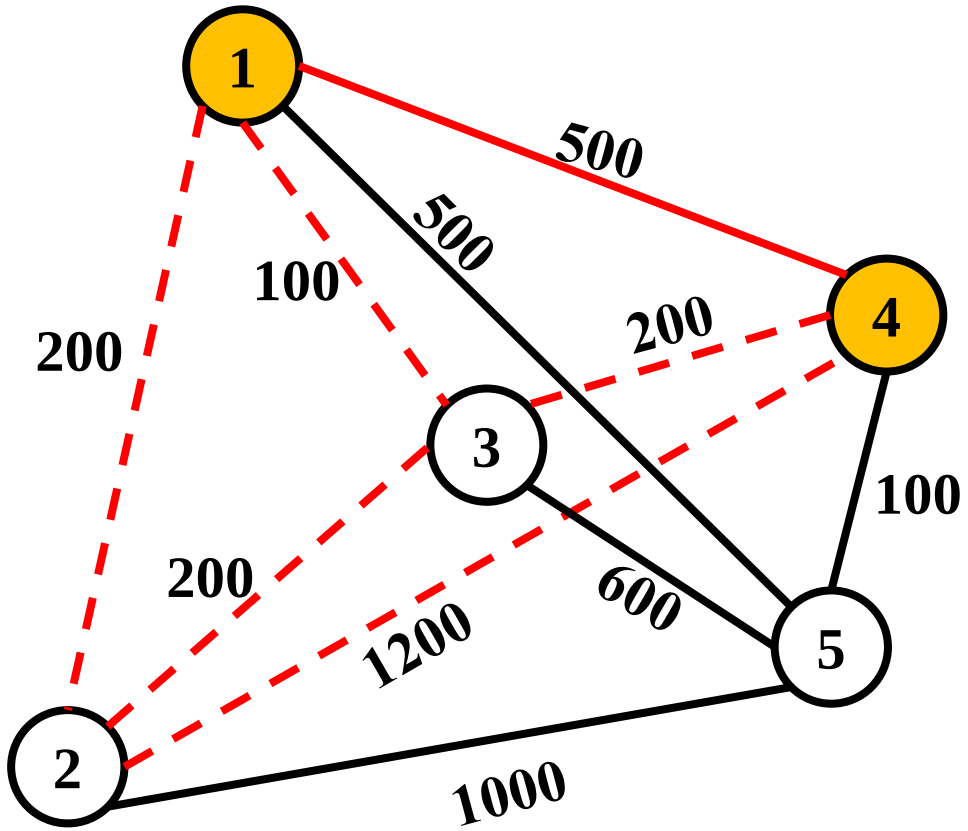
递推关系建立

自底向上计算

最优方案追踪

动态规划：递推关系建立

- 如果不选第k个点经过
  - $D[k,i,j]=D[k-1,i,j]$



本例中

$k=2,i=1,j=4$

$D[2,1,4]=D[1,1,4]=500$

问题结构分析

递推关系建立

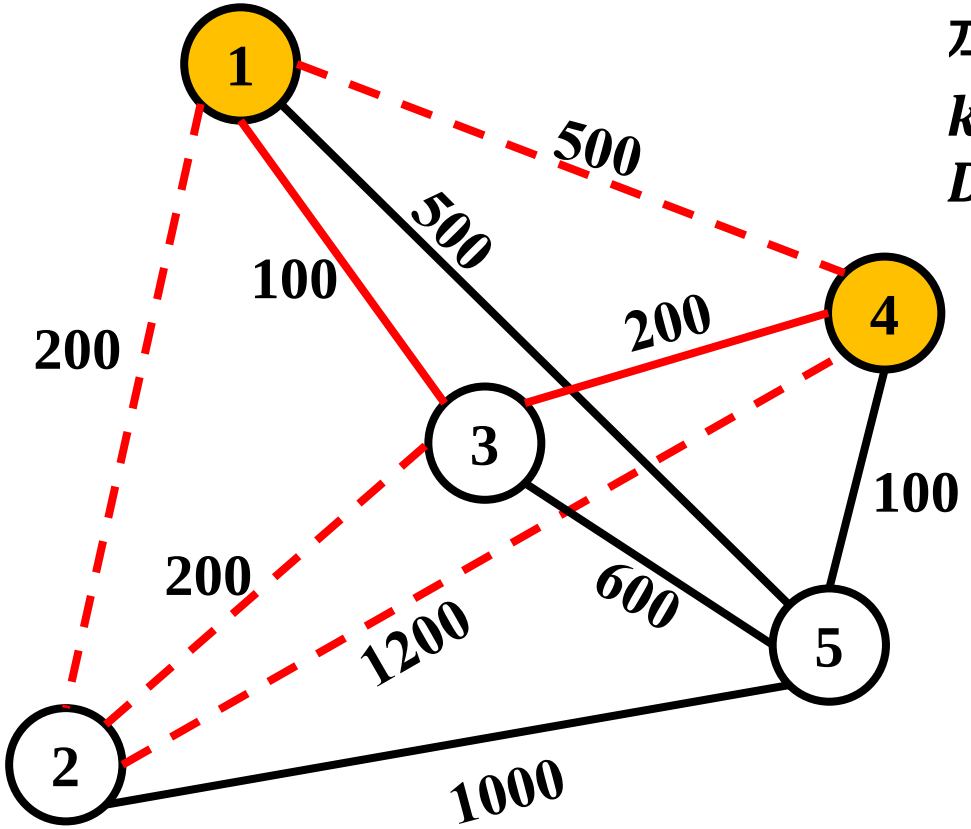
自底向上计算

最优方案追踪

动态规划：递推关系建立

- 如果不选第k个点经过
  - $D[k,i,j]=D[k-1,i,j]$
- 如果选择第k个点经过
  - $D[k,i,j]=D[k-1,i,k]+D[k-1,k,j]$

表示松弛成功



本例中

$k=3,i=1,j=4$

$D[3,1,4]=D[2,1,3]+D[2,3,4]$

$=100+200=300$

问题结构分析

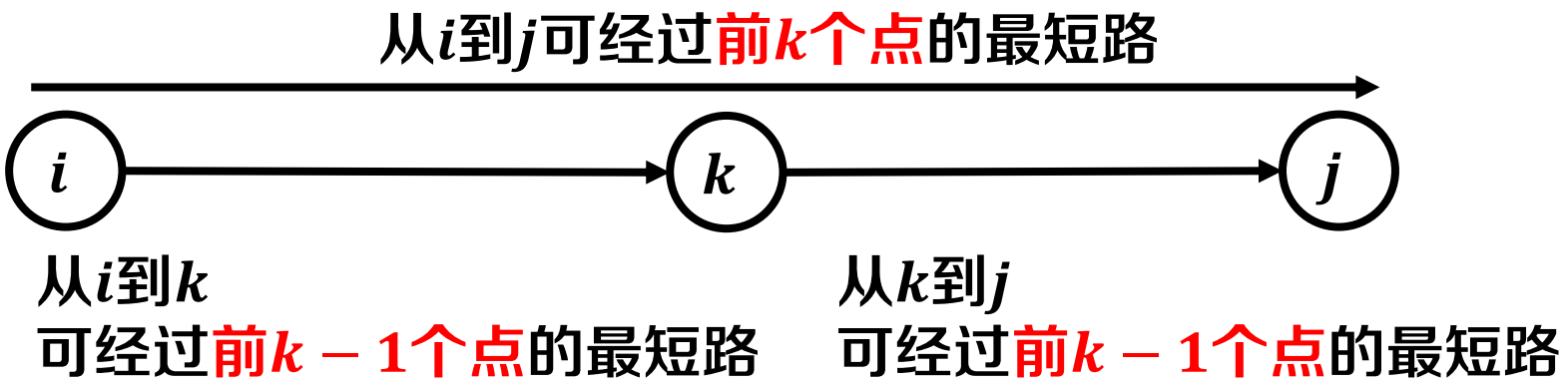
递推关系建立

自底向上计算

最优方案追踪

动态规划：递推关系建立

- 如果不选第k个点经过
  - $D[k,i,j]=D[k-1,i,j]$
- 如果选择第k个点经过
  - $D[k,i,j]=D[k-1,i,k]+D[k-1,k,j]$



问题结构分析

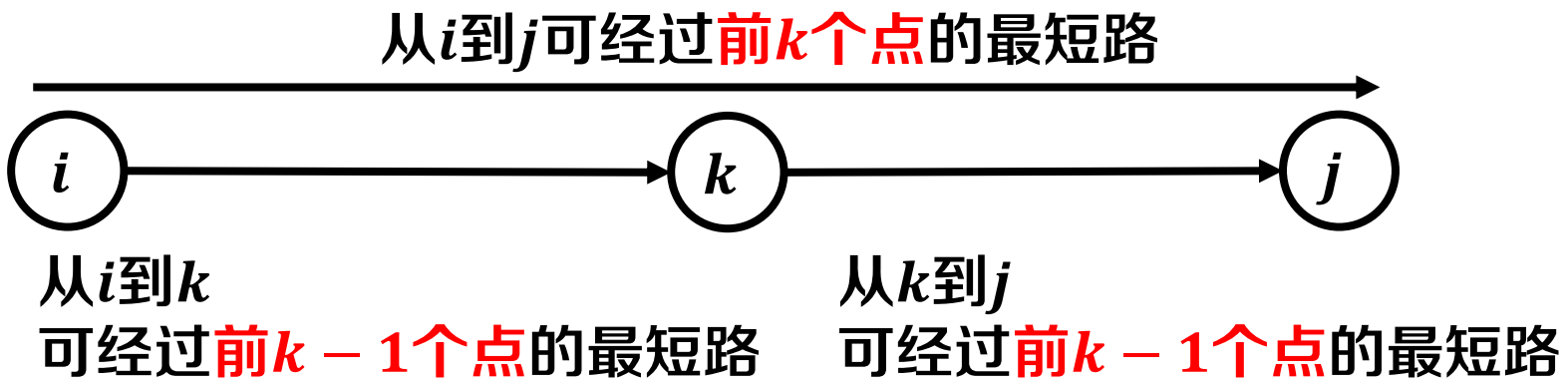
递推关系建立

自底向上计算

最优方案追踪

动态规划：递推关系建立

- 如果不选第k个点经过
  - $D[k,i,j]=D[k-1,i,j]$
- 如果选择第k个点经过
  - $D[k,i,j]=D[k-1,i,k]+D[k-1,k,j]$



- $D[k,i,j]=\min\{D[k-1,i,j],$
  $D[k-1,i,k]+D[k-1,k,j]\}$

问题结构分析

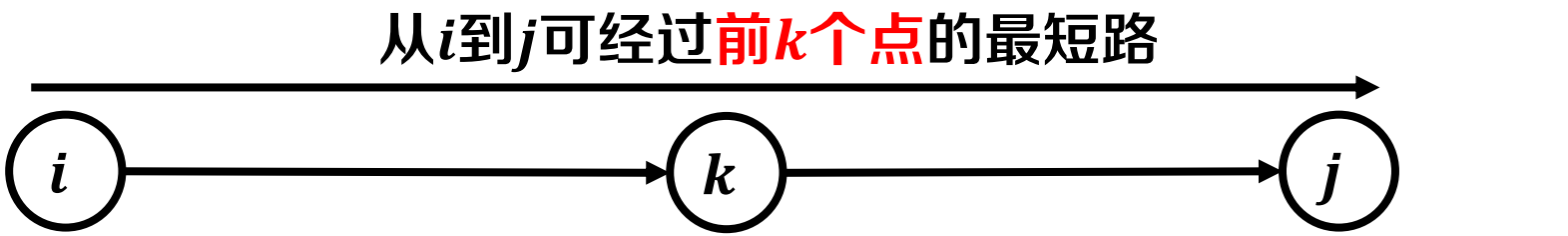
递推关系建立

自底向上计算

最优方案追踪

动态规划：递推关系建立

- 如果不选第k个点经过
  - $D[k,i,j]=D[k-1,i,j]$
- 如果选择第k个点经过
  - $D[k,i,j]=D[k-1,i,k]+D[k-1,k,j]$



- $D[k,i,j]=\min\{D[k-1,i,j],$
  $D[k-1,i,k]+D[k-1,k,j]\}$

最优子结构

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

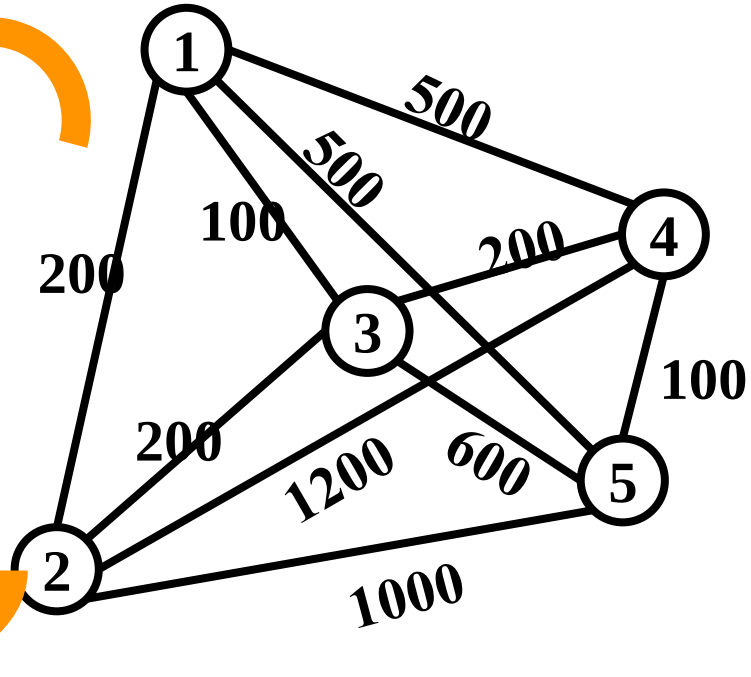
- 初始化
  - $D[0,i,i]=0$：起终点重合，路径长度为0

i \ j	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

动态规划：自底向上计算（确定计算顺序）

- 初始化
  - $D[0,i,i]=0$：起终点重合，路径长度为0
  - $D[0,i,j]=e[i,j]$：任意两点直达距离为边权

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0



问题结构分析

递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j]=\min\{D[k-1,i,j],$
    $D[k-1,i,k]+D[k-1,k,j]\}$

最终的表格：$k=|V|$

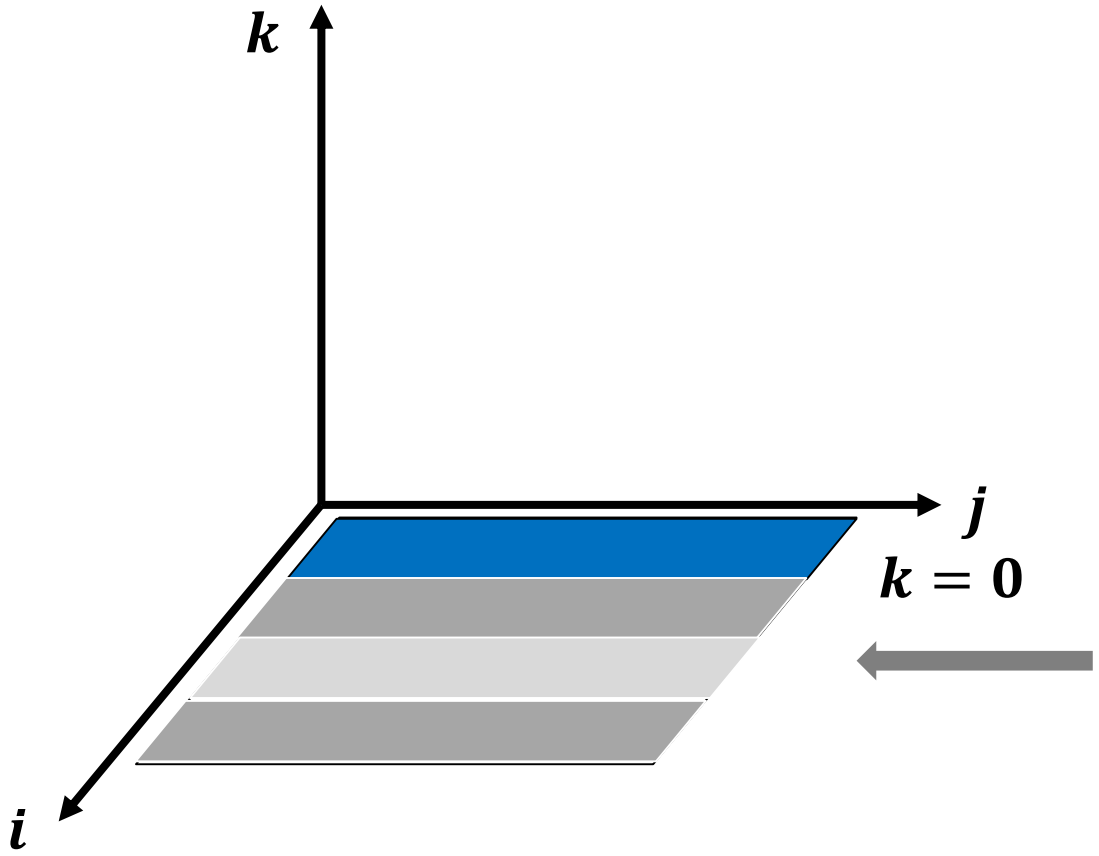
i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k=0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$



最终的表格：$k = |V|$

i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

问题结构分析

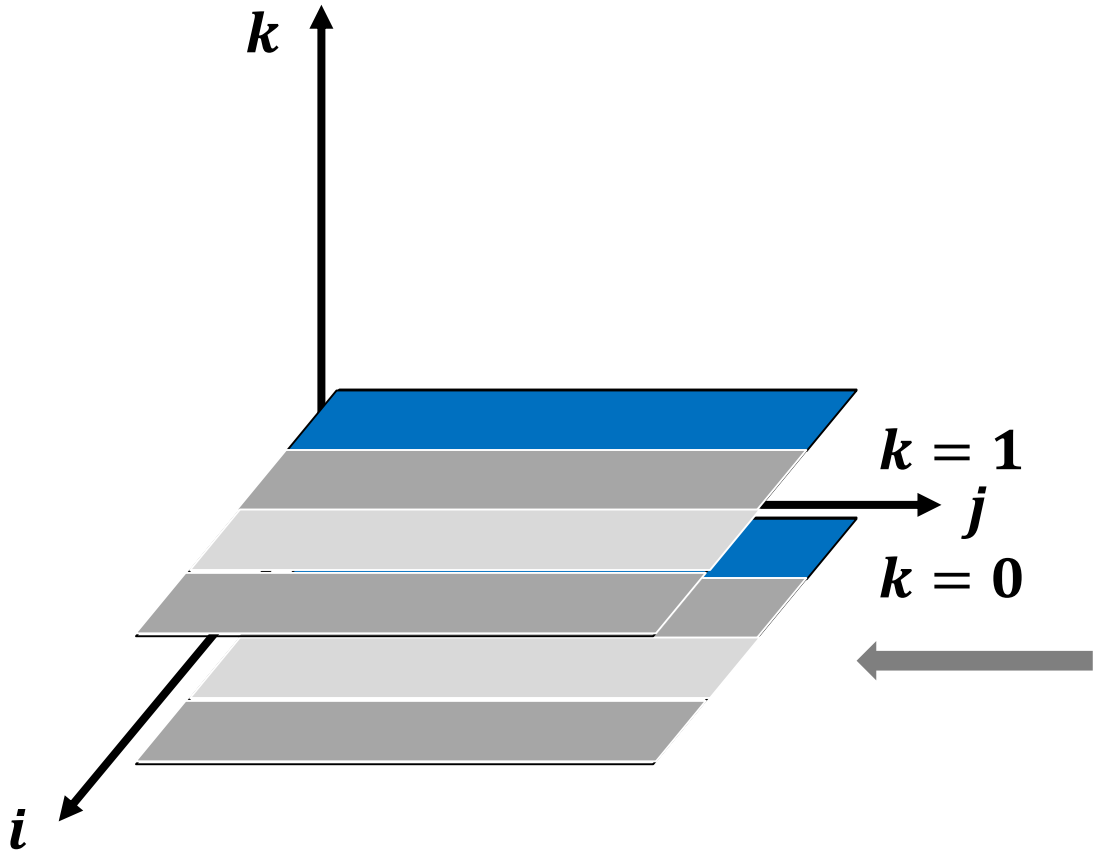
递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j], D[k-1,i,k] + D[k-1,k,j]\}$



最终的表格：$k = |V|$

i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

问题结构分析

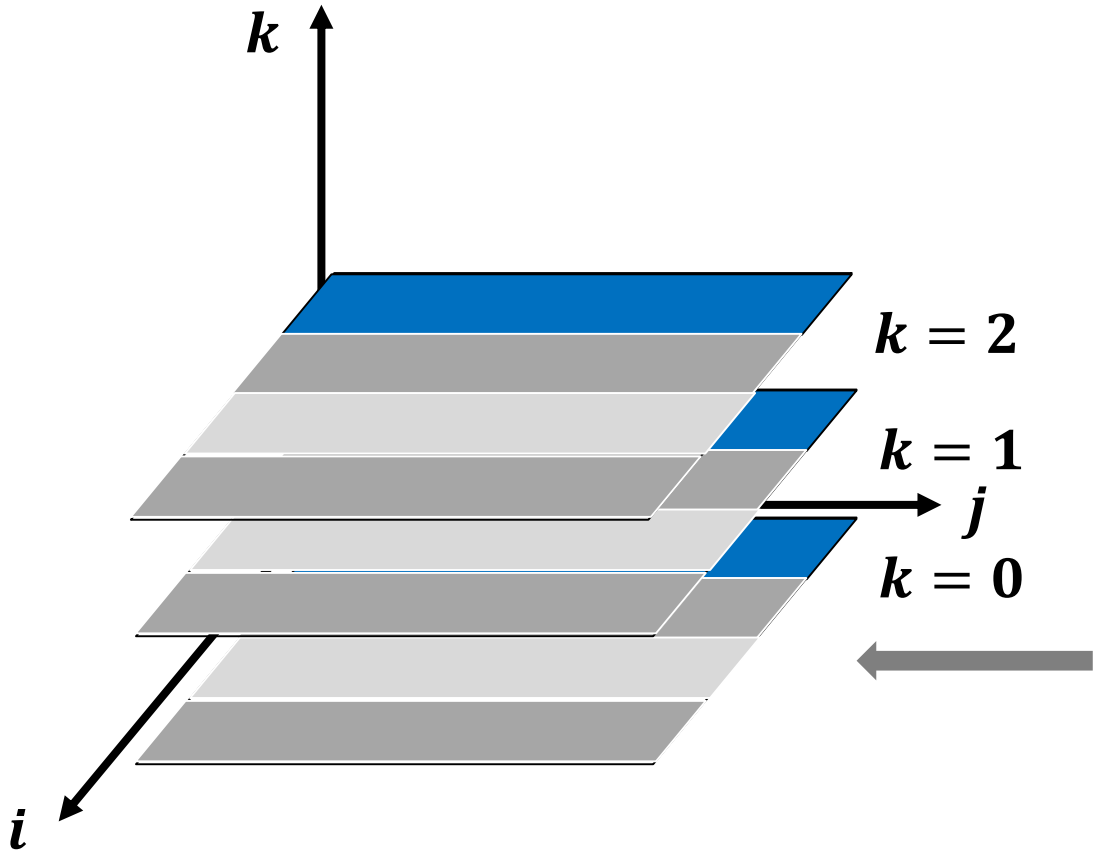
递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j],$
    $D[k-1,i,k] + D[k-1,k,j]\}$



最终的表格：$k = |V|$

i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k = 0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

问题结构分析

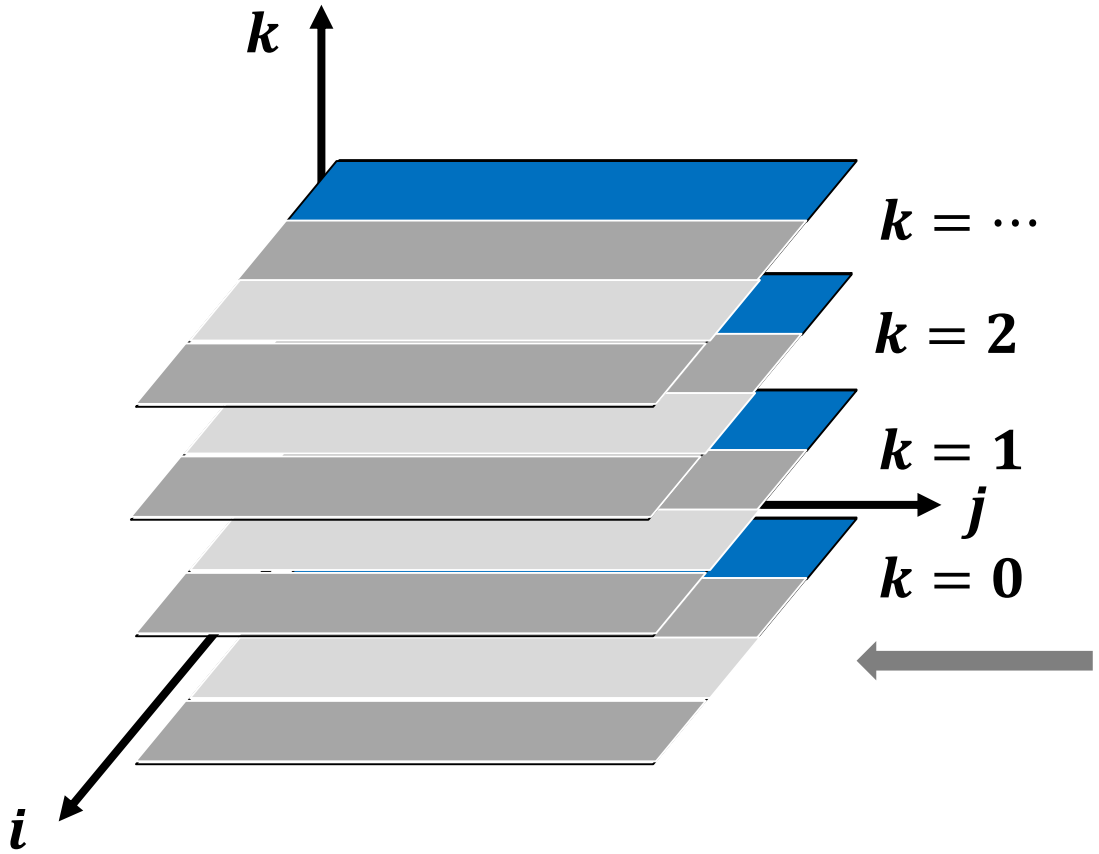
递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j]=\min\{D[k-1,i,j],$
    $D[k-1,i,k]+D[k-1,k,j]\}$



最终的表格：$k=|V|$

i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k=0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

问题结构分析

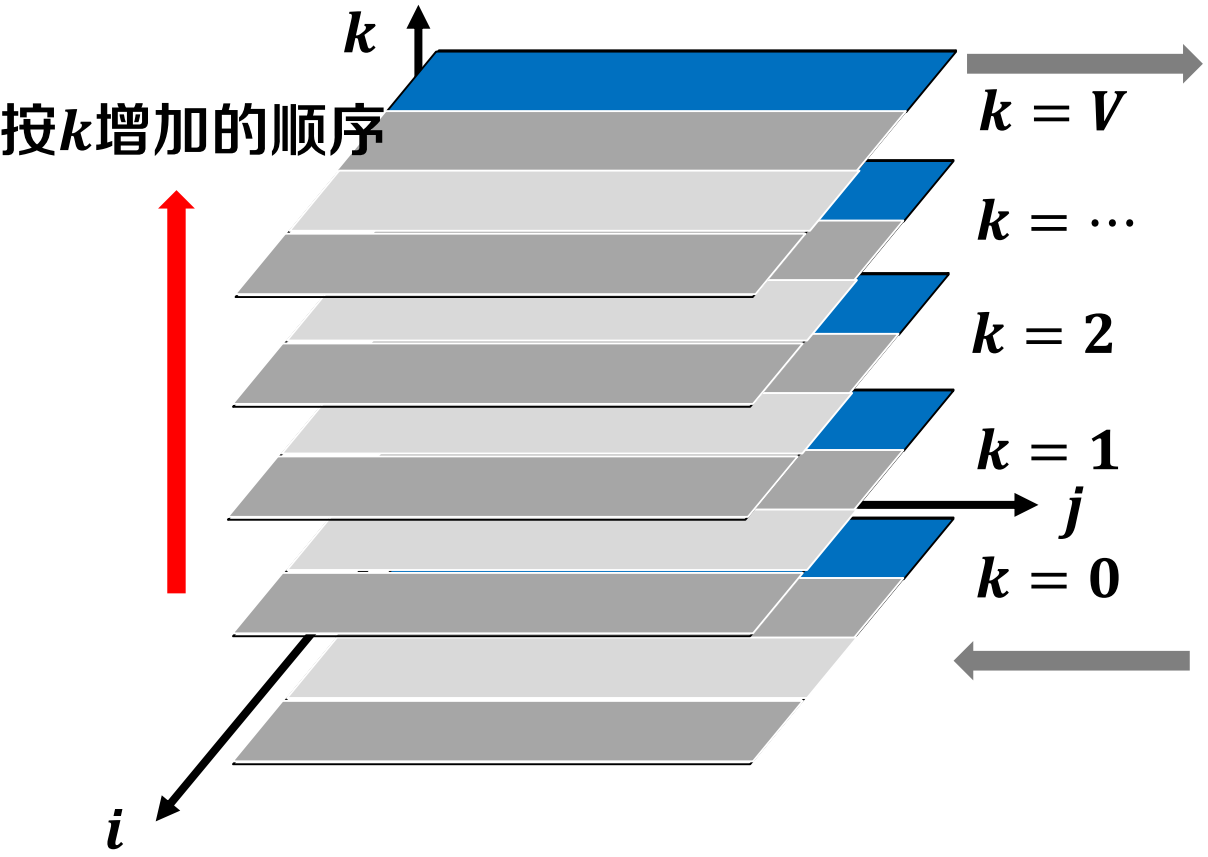
递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j]=\min\{D[k-1,i,j],$
    $D[k-1,i,k]+D[k-1,k,j]\}$



最终的表格：$k=|V|$

i \ j	1	…	…	…	…
1					
…			?		
…					
…					
…					

初始化的表格：$k=0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

问题结构分析

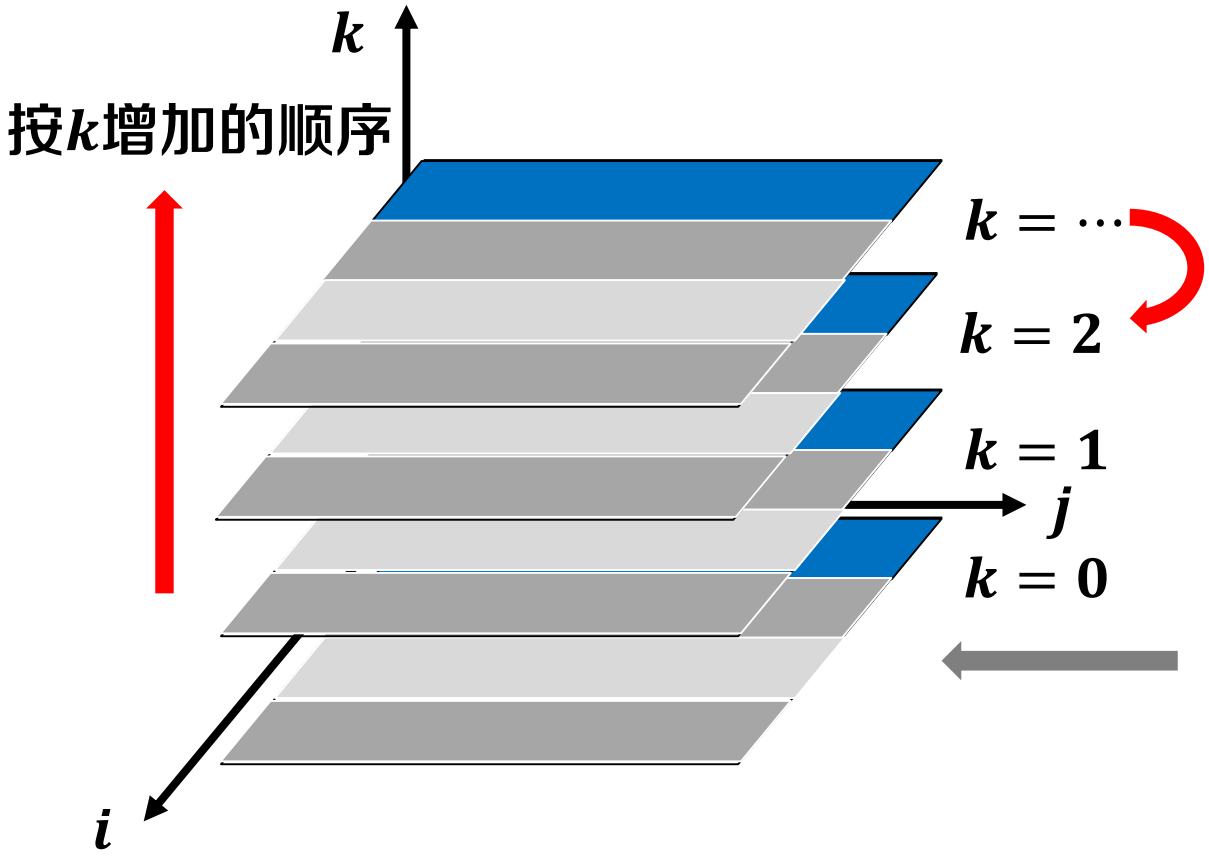
递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j]=\min\{D[k-1,i,j],$
    $D[k-1,i,k]+D[k-1,k,j]\}$



求解当前层只依赖上一层

初始化的表格：$k=0$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

问题结构分析

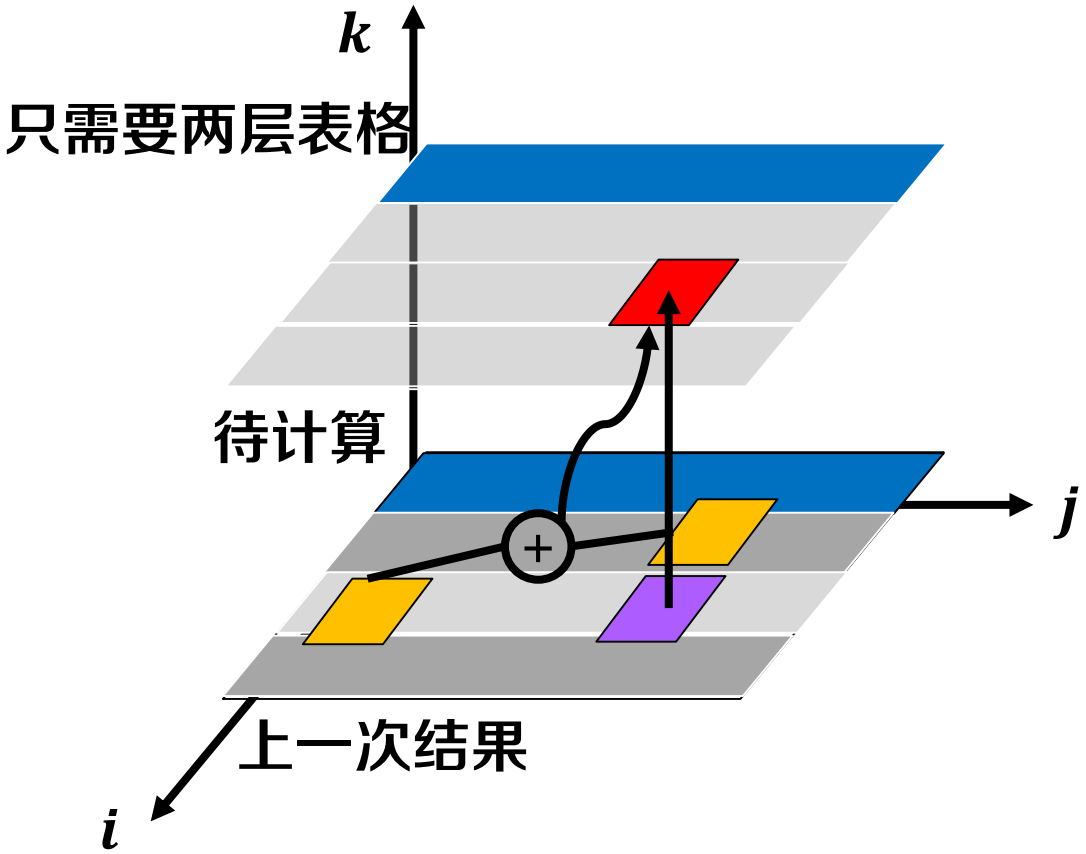
递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j],$
    $D[k-1,i,k] + D[k-1,k,j]\}$



问题结构分析

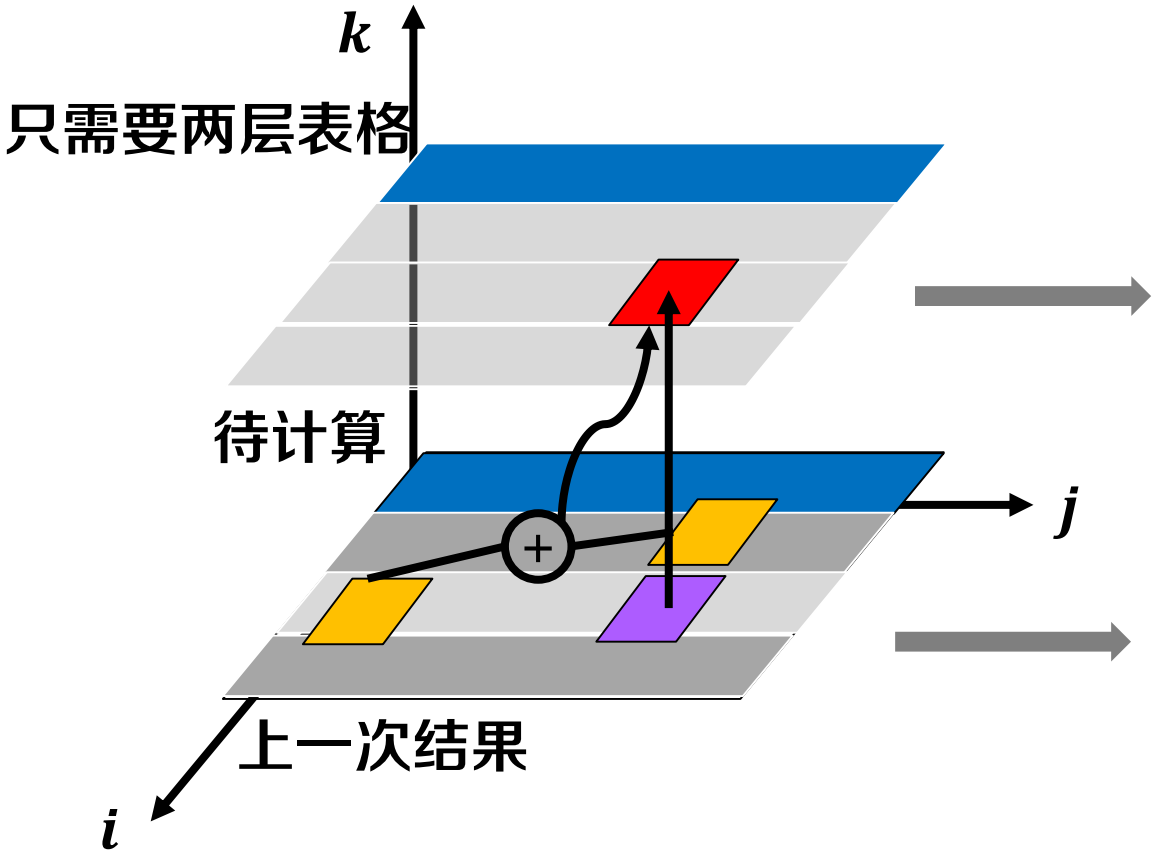
递推关系建立

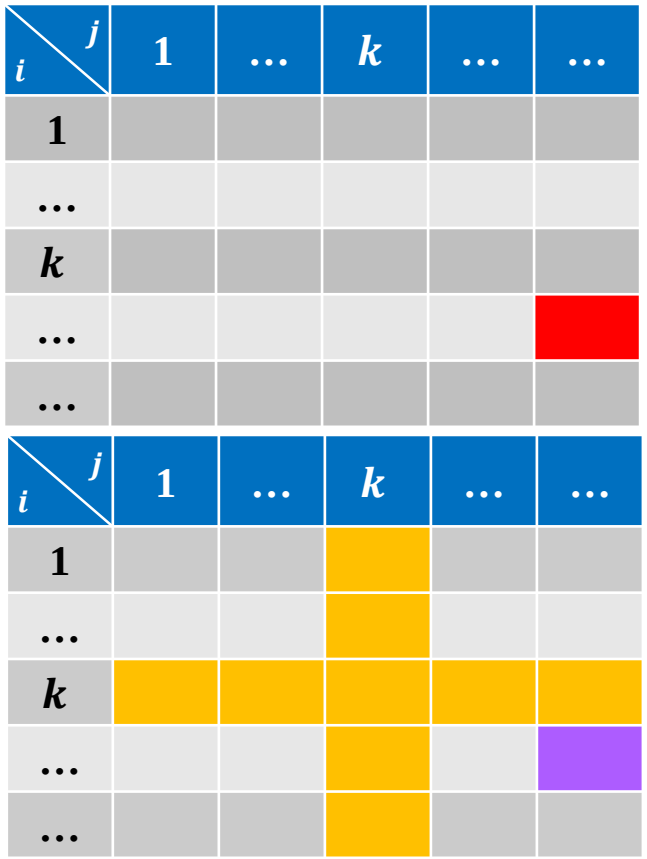
自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j],$
    $D[k-1,i,k] + D[k-1,k,j]\}$





问题结构分析

递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j] = \min\{D[k-1,i,j],$ $D[k-1,i,k] + D[k-1,k,j]\}$
    $0 + D[k-1,i,j] = D[k-1,i,j]$
- 若$k = i$或$k = j$

$$D[k,i,j] = D[k-1,i,j]$$

值相同，可以直接覆盖

i \ j	1	…	k	…	…
1					
…					
k					
…					
…					

i \ j	1	…	k	…	…
1					
…					
k					
…					
…					

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j]=\min\{D[k-1,i,j],$ $D[k-1,i,k]+D[k-1,k,j]\}$

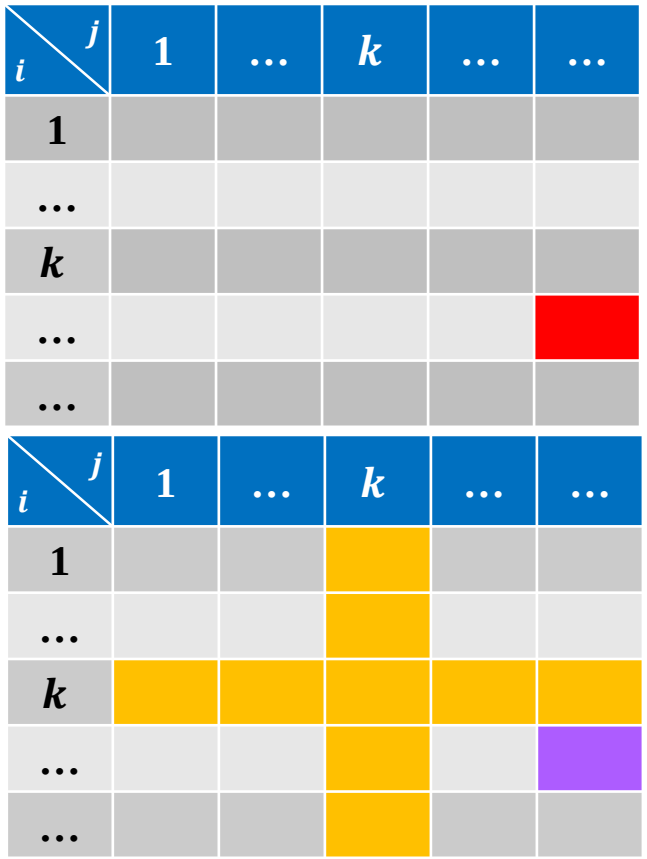
$0+D[k-1,i,j]=D[k-1,i,j]$

- 若$k=i$或$k=j$

$D[k,i,j]=D[k-1,i,j]$
值相同，可以直接覆盖

- 若$k\neq i$且$k\neq j$

$D[k-1,i,j]$和$D[k-1,i,k],D[k-1,k,j]$
不是相同子问题
求出$D[k,i,j]$后，$D[k-1,i,j]$不再被使用
可直接覆盖



问题结构分析

递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D[k,i,j]=\min\{D[k-1,i,j],$ $D[k-1,i,k]+D[k-1,k,j]\}$

$0+D[k-1,i,j]=D[k-1,i,j]$

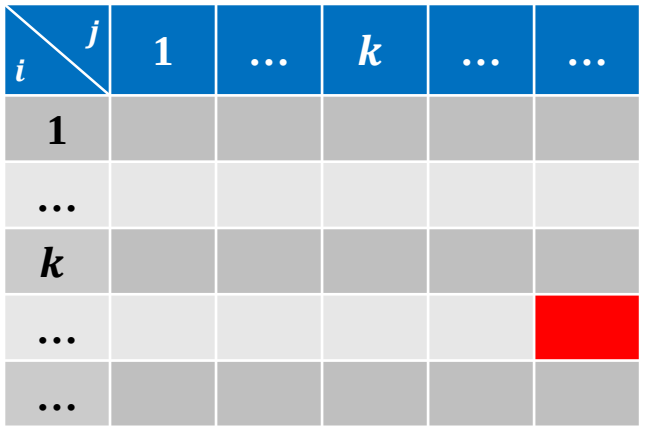
- 若$k=i$或$k=j$

$D[k,i,j]=D[k-1,i,j]$
值相同，可以直接覆盖

- 若$k\neq i$且$k\neq j$

$D[k-1,i,j]$和$D[k-1,i,k],D[k-1,k,j]$
不是相同子问题
求出$D[k,i,j]$后，$D[k-1,i,j]$不再被使用
可直接覆盖

求出新值可直接在原位置覆盖
只需存储一层表格



问题结构分析

递推关系建立

自底向上计算

最优方案追踪

动态规划：自底向上计算（确定计算顺序）

- 递推公式
  - $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

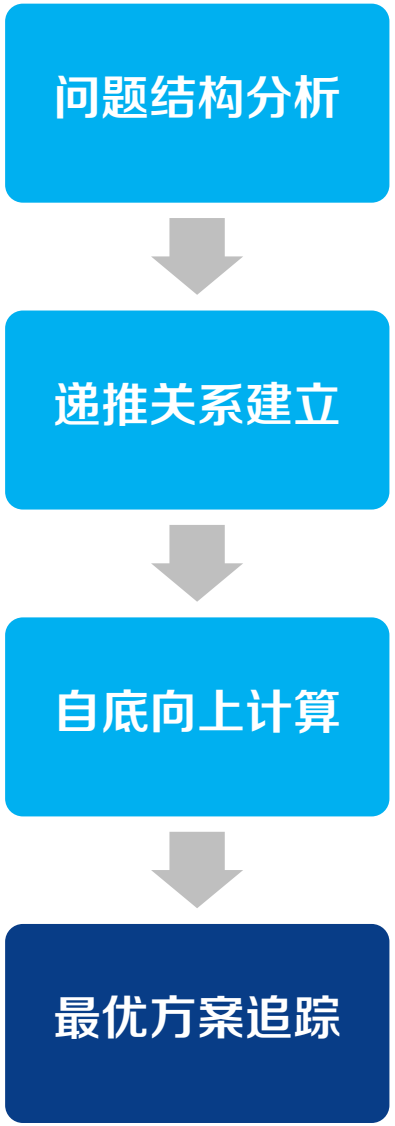
求出新值可直接在原位置覆盖
只需存储一层表格

动态规划：最优方案追踪

- 递推公式
  - $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 追踪数组Rec，记录经过的中间点
  - $D_k[i,j] = D_{k-1}[i,j]$：0 表示没有中间点

Rec

i \ j	1	…	j	…	V
1					
…					
i			0		
…					
V					



动态规划：最优方案追踪

- 递推公式
  - $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 追踪数组Rec，记录经过的中间点
  - $D_k[i,j] = D_{k-1}[i,j]$：0 表示没有中间点
  - $D_k[i,j] = D_{k-1}[i,k] + D_{k-1}[k,j]$：$k$ 表示经过中间点k

松弛时使用的点

Rec

i \ j	1	…	j	…	V
1					
…					
i			k		
…					
V					

问题结构分析

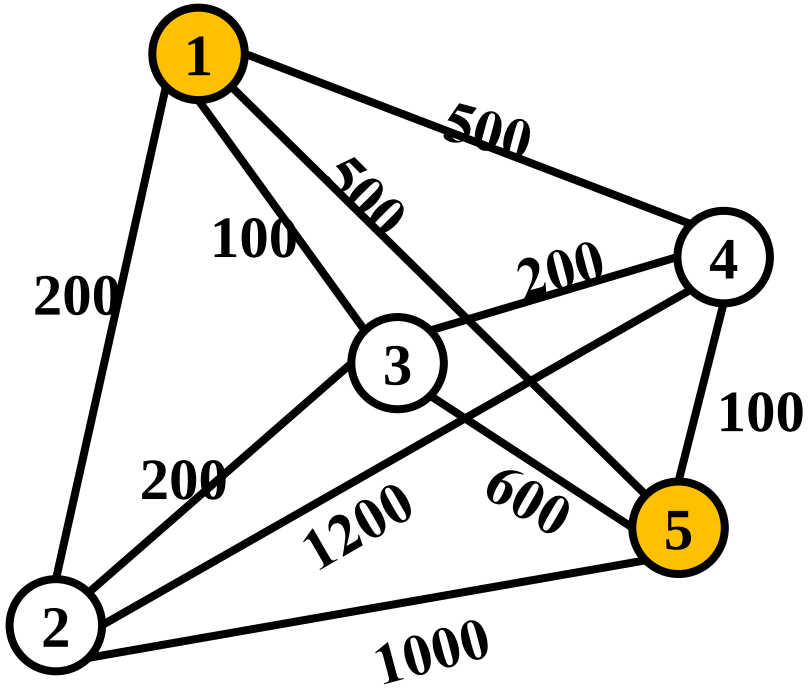
递推关系建立

自底向上计算

最优方案追踪

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

问题结构分析

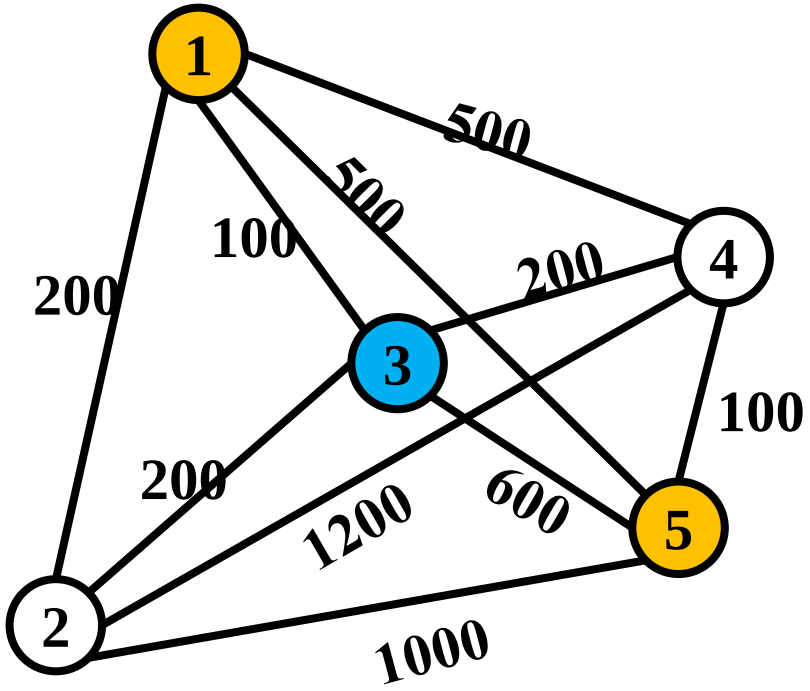
递推关系建立

自底向上计算

最优方案追踪

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

问题结构分析

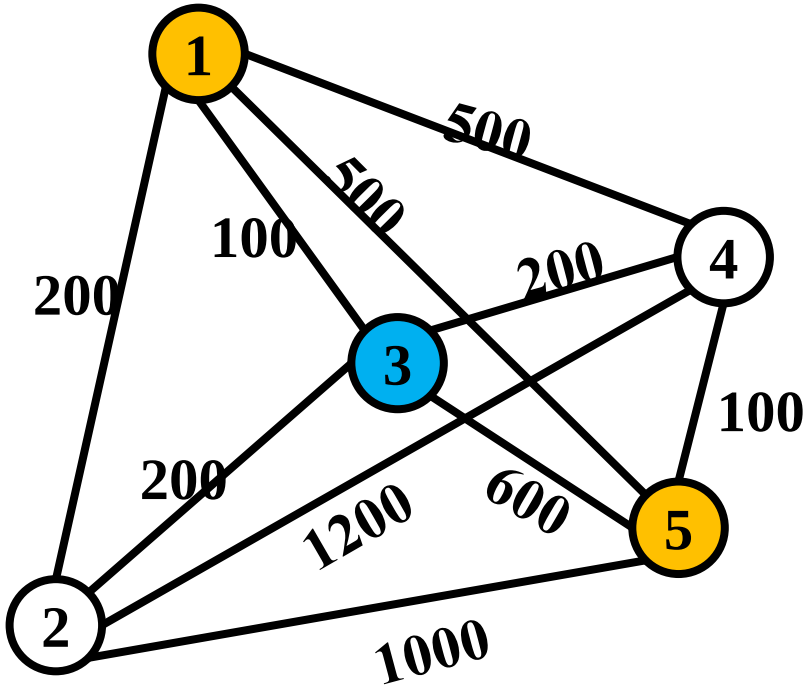
递推关系建立

自底向上计算

最优方案追踪

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

问题结构分析

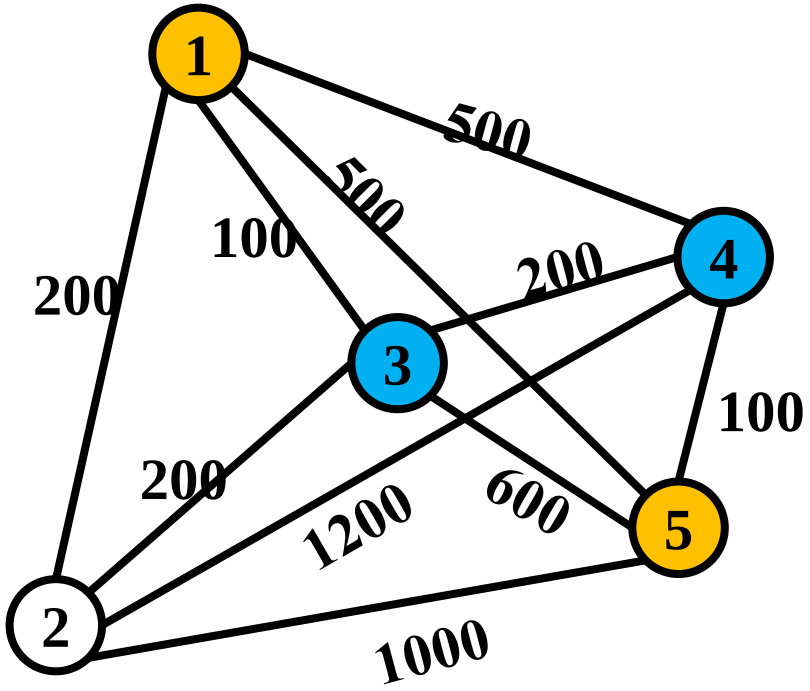
递推关系建立

自底向上计算

最优方案追踪

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

问题结构分析

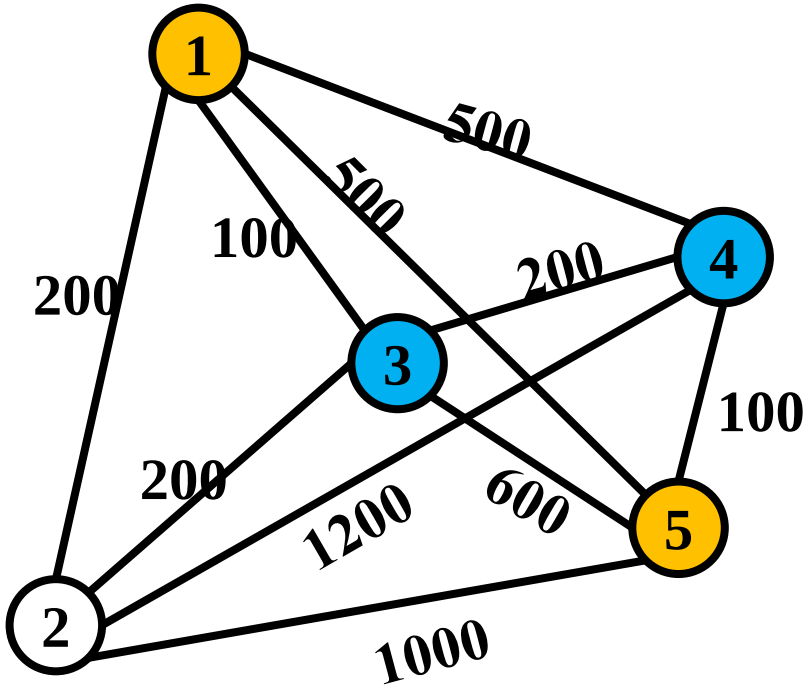
递推关系建立

自底向上计算

最优方案追踪

动态规划：最优方案追踪

- 根据数组Rec，输出最短路径



Rec

i \ j	1	2	3	4	5
1			0		3
2					
3				0	4
4					0
5					

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

提纲

问题定义

算法思想

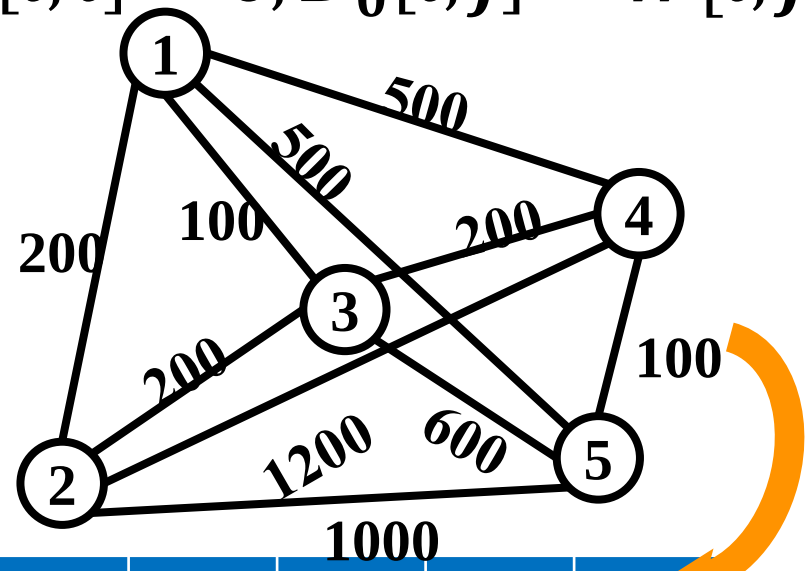
算法设计

算法实例

算法分析

算法实例

- $D_0[i,i]=0, D_0[i,j]=W[i,j]$



D

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

$k=0$

Rec

所有点对都没有经过其他点

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

D

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	1200	1000
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

$k = 1$

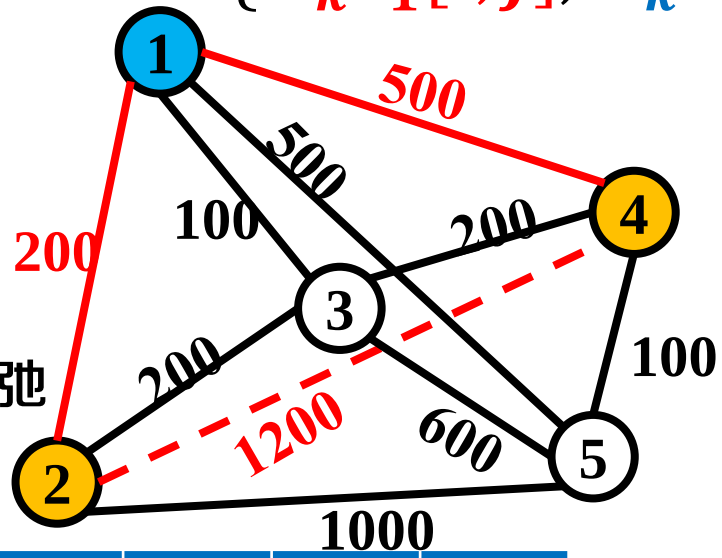
Rec

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

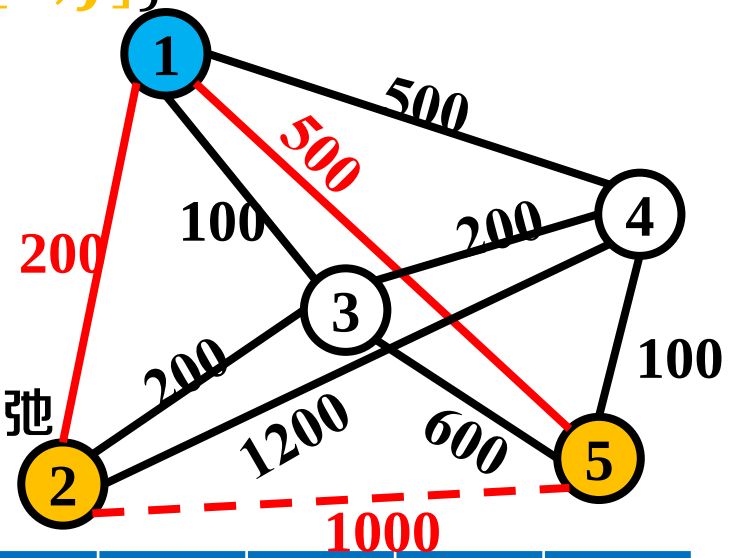
算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

从2到4路径得到松弛



从2到5路径得到松弛



D

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

$k = 1$

Rec

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	1200	200	0	100
5	500	1000	600	100	0

$k = 1$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	1000	600	100	0

$k = 1$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 1$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

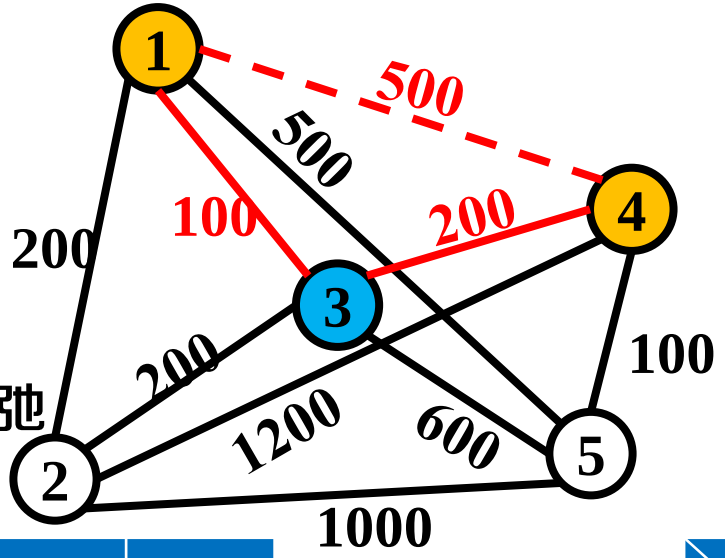
i \ j	1	2	3	4	5
1	0	200	100	500	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 2$

i \ j	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$



从1到4路径得到松弛

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	700	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	400	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	3	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	400	700
3	100	200	0	200	600
4	500	700	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	3	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	400	700
3	100	200	0	200	600
4	300	400	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	3	1
3	0	0	0	0	0
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	500
2	200	0	200	400	700
3	100	200	0	200	600
4	300	400	200	0	100
5	500	700	600	100	0

$k = 3$

i \ j	1	2	3	4	5
1	0	0	0	3	0
2	0	0	0	3	1
3	0	0	0	0	0
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	700
3	100	200	0	200	600
4	300	400	200	0	100
5	500	700	600	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	1
3	0	0	0	0	0
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	600
4	300	400	200	0	100
5	500	700	600	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	0
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	500	700	600	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	500	700	600	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	0	0	0	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 4$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$

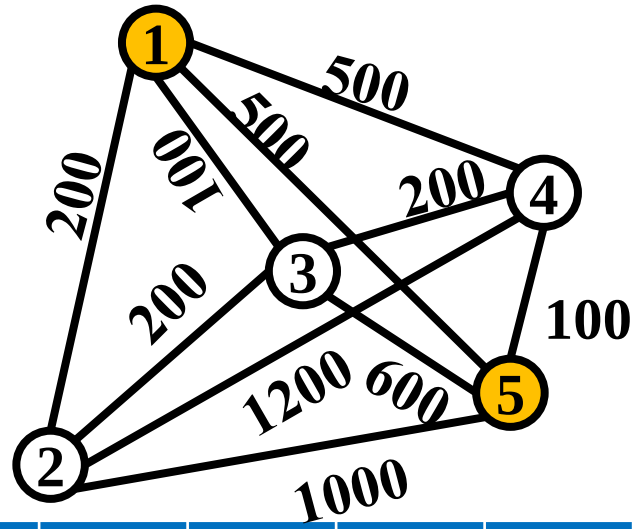
i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

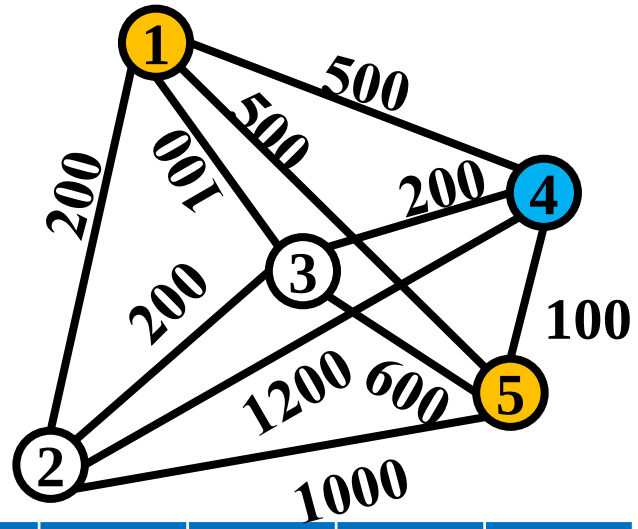
$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

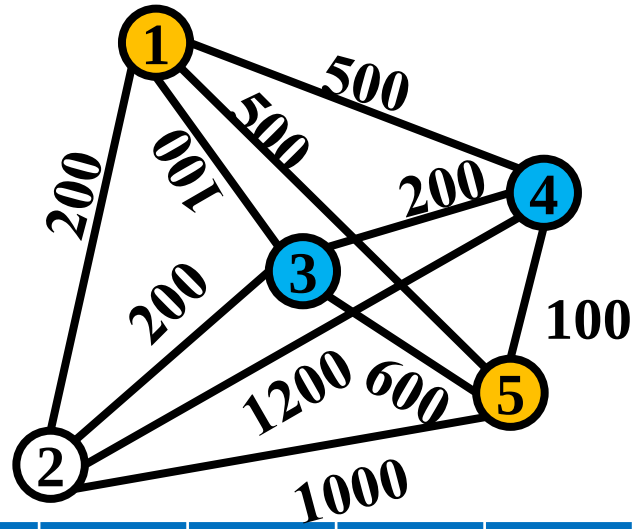
$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

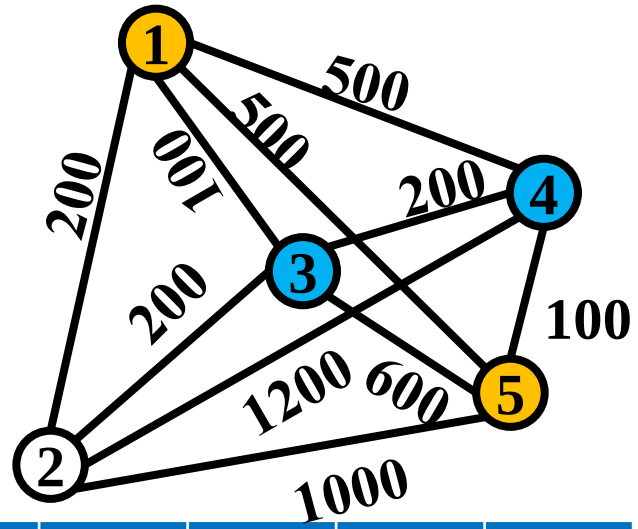
$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

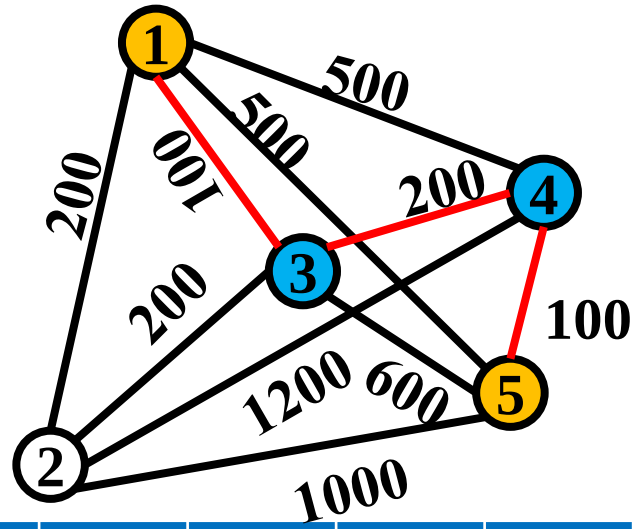
$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

算法实例

- $D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}$
- 查询从1到5的最短路



D

i \ j	1	2	3	4	5
1	0	200	100	300	400
2	200	0	200	400	500
3	100	200	0	200	300
4	300	400	200	0	100
5	400	500	300	100	0

$k = 5$

Rec

i \ j	1	2	3	4	5
1	0	0	0	3	4
2	0	0	0	3	4
3	0	0	0	0	4
4	3	3	0	0	0
5	4	4	4	0	0

提纲

问题定义

算法思想

算法设计

算法实例

算法分析

伪代码

- **All-Pairs-Shortest-Paths(G)**

输入: 图$G =< V, E, W >$
输出: 任意两点最短路径
新建二维数组$D[1..V, 1..V], Rec[1..V, 1..V]$

```
for i ← 1 to V do
    for j ← 1 to V do
        Rec[i, j] ← 0
        if i = j then
            D[i, j] ← 0
        end
        else
            D[i, j] ← W[i, j]
        end
    end
end
```

初始化

伪代码

- **All-Pairs-Shortest-Paths(G)**

```
输入: 图G =< V, E, W >
输出: 任意两点最短路径
新建二维数组D[1..V, 1..V], Rec[1..V, 1..V]
for i ← 1 to V do
    for j ← 1 to V do
        Rec[i, j] ← 0
        if i = j then
            D[i, j] ← 0
        end
        else
            D[i, j] ← W[i, j]
        end
    end
end
```

起终点相同，距离为0

起终点不同，距离为边权

伪代码

- **All-Pairs-Shortest-Paths(G)**

```
for k ← 1 to V do
    for i ← 1 to V do
        for j ← 1 to V do
            if D[i, j] > D[i, k] + D[k, j] then
                D[i, j] ← D[i, k] + D[k, j]
                Rec[i, j] ← k
            end
        end
    end
end
return D, Rec
```

按照k增大的顺序

伪代码

- **All-Pairs-Shortest-Paths(G)**

```
for k ← 1 to V do
    for i ← 1 to V do
        for j ← 1 to V do
            if D[i, j] > D[i, k] + D[k, j] then
                D[i, j] ← D[i, k] + D[k, j]
                Rec[i, j] ← k
            end
        end
    end
end
return D, Rec
```

松弛操作

伪代码

- **Find-Path(Rec, u, v)**

输入: 备忘数组Rec,起点u,终点v
输出: 最短路径(逆序)
if $Rec[u, v] = 0$ then
    print v
    return
end
$k \leftarrow Rec[u, v]$
Find-Path(Rec, u, k)
Find-Path(Rec, k, v)

没有中间点，直接输出

伪代码

- **Find-Path(Rec, u, v)**

输入: 备忘数组Rec,起点u,终点v
输出: 最短路径(逆序)
if $Rec[u, v] = 0$ then
    print v
    return
end
$k \leftarrow Rec[u, v]$
Find-Path(Rec, u, k)
Find-Path(Rec, k, v)

有中间点，递归查找

时间复杂度

- **All-Pairs-Shortest-Paths(G)**

```
for k ← 1 to V do
    for i ← 1 to V do
        for j ← 1 to V do
            if D[i, j] > D[i, k] + D[k, j] then
                D[i, j] ← D[i, k] + D[k, j]
                Rec[i, j] ← k
            end
        end
    end
end
return D, Rec
```

$O(|V|)$ $O(|V|^2)$ $O(|V|^3)$

时间复杂度：$O(|V|^3)$

算法小结

- 该算法由Floyd和Warshall于1962年分别提出
- 也被称为Floyd-Warshall算法

Robert Floyd
1936-2001

Stephen Warshall
1935-2006

算法小结

- 直观思路：使用Dijkstra算法依次求解所有点

```
输入: 图G
输出: 任意两点最短路径
for i ← 1 to V do
    Paths[i] ← Dijkstra − PriQueue(G, i)
end
return Paths
```

$O(|E|\log|V|)$

$O(|V||E|\log|V|)$

回顾

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

时间复杂度$O(|E| \cdot \log|V|)$

算法小结

- **直观思路：使用Dijkstra算法依次求解所有点**

输入: 图G
输出: 任意两点最短路径
for $i \leftarrow 1$ *to* V **do**
  $Paths[i] \leftarrow Dijkstra - PriQueue(G, i)$
end
return *Paths*

$O(|E|\log|V|)$ → $O(|V||E|\log|V|)$

- **Floyd-Warshall算法时间复杂度：$O(|V|^3)$**

算法小结

- 直观思路：使用Dijkstra算法依次求解所有点

输入: 图G
输出: 任意两点最短路径
for $i \leftarrow 1$ *to* V **do**
| $Paths[i] \leftarrow Dijkstra - PriQueue(G, i)$
end
return *Paths*

$O(|E|\log|V|)$

$O(|V||E|\log|V|)$

图中边较多时
$|E| = O(|V|^2)$

$O(|V|^3\log|V|)$

- Floyd-Warshall算法时间复杂度：$O(|V|^3)$

算法小结

- 直观思路：使用Dijkstra算法依次求解所有点

输入: 图G
输出: 任意两点最短路径
for $i \leftarrow 1$ *to* V **do**
| $Paths[i] \leftarrow Dijkstra - PriQueue(G, i)$
end
return *Paths*

$O(|E|\log|V|)$

$O(|V||E|\log|V|)$

图中边较多时
$|E| = O(|V|^2)$

- Floyd-Warshall算法时间复杂度：$O(|V|^3)$ 优于 $O(|V|^3\log|V|)$

最短路径算法小结

- 最短路径算法
  - 单源最短路径算法
    - 广度优先搜索（适用于边权都为1的情况）
      $O(|V|+|E|)$
    - Dijkstra算法（适用于边权全非负的情况）
      $O(|E|\cdot \log|V|)$
    - Bellman-Ford算法（适用于边权可以为负的情况）
      $O(|E|\cdot|V|)=O(|V|^3)$（当边较多时$|E|=O(|V|^2)$）
  - 所有点对最短路径算法 Floyd-Warshall算法（适用于边权可以为负的情况）
    $O(|V|^3)$

谢谢

BDA
Beihang University