Ch6 动态规划

1. 基本概念

1.1 介绍

通过细心的把事情分解为一系列子问题，然后对越来越大的子问题建立正确的解，从而隐含地探查所有可行解的空间

1.2 应用

领域

- 生物信息学
- 控制理论
- 信息理论
- 操作研究
- 计算机科学：理论，图形，人工智能，编译器，系统

一些著名的动态规划算法

- Unix diff 比较两个文件
- Viterbi 隐藏马尔科夫模型
- Smith-Waterman 基因序列比对
- Bellman-Ford 最短路径路由网络
- Cocke-Kasami-Younger 用于解析上下文自由语法

1.3 备忘录或者子问题的迭代

请先阅读**带权区间调度**部分，然后回来看本节

备忘录

我们开发了一个带权区间问题在多项式时间内的解法，它先设计了一个指数时间的递归算法，然后通过备忘录的形式，把它转化成了一个有效的递归算法，这个算法考虑**对于子问题最优解全局数组** `OPT[]`

实际上，有一个与这个算法基本等价的方法，同时为我们开发了动态规划算法的一般模板

子问题迭代

有效算法的关键实际上是数组 `OPT[]`，我们的问题其实变成了

求出 `OPT[]`，`OPT[n]` 包含了整个问题最优解的值，然后 `Find-Solution()` 通过 `OPT[]` 有效的追踪并且返回最优解本身

我们可以不使用备忘录式的递归，而是直接迭代计算 `OPT[]` 的各项，从 `OPT[0] = 0` 开始

```
Iterative-OPT:
    把工件按照结束时间 f1 ≤ f2 ≤ ... ≤ fn 进行排列;
    计算 p(1),p(2),...p(n);
    OPT[0] = 0;
    for(j = 1,2,...,n){
        OPT[j] = max(v_j + OPT[p(j)], OPT[j-1])
    }
```

接下来的部分，我们多使用这种迭代类型的方法——建立在子问题上的迭代，来开发动态规划算法

当然，迭代算法和备忘录算法是可以互相转化的

1.4 动态规划的模板

人们需要从一组初始问题中导出满足某些基本性质的子问题

- 只存在多项式个子问题
- 可以很容易的从子问题的解计算初始问题的解
- 在子问题中从“最小”到“最大”存在一种自然的关系，与一个容易计算的递推式相联系，这个递推式允许人们从某些更小的子问题来确定一个子问题的解

动态规划的基本解题四路

- 问题结构的表征
- 递归定义最优解的值
- 计算最优解的值
- 从求解的信息返回最优解

动态规划的解题方式

- 从后向前计算 OPT
- 从前向后计算 OPT

2. 带权区间调度 Weighted Interval Scheduling

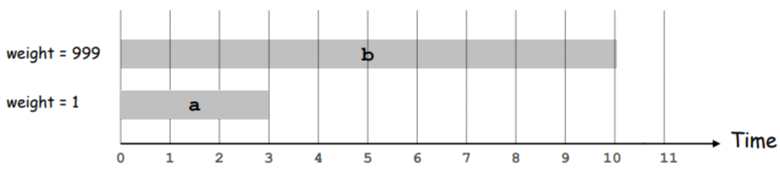
2.1 问题描述

任务 j 的开始时间为 s_j，结束时间为 f_j，它有一个价值 v_j

两个工作如果不重叠的话是兼容的

目标：找到相互兼容的任务且满足总价值最大

我们在贪心算法部分也介绍了区间调度问题，不过当时每个区间的权重是 1，在带权区间问题中，每个区间拥有一个权重，如果允许任意权值，贪心算法就会失败



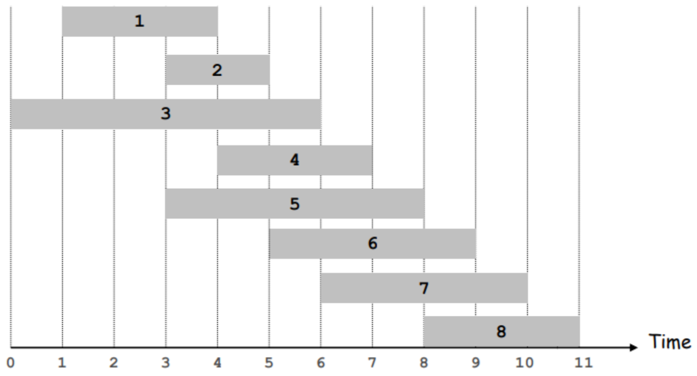
- 贪婪算法会选择 a

2.2 算法

首先我们按照结束时间排列每个区间，使得 $f_1 \le f_2 \le \ldots \le f_n$

定义 $p(j)$：最大的区间编号 i 使得 $i < j$，且 i 和 j 是相容的

p(8) = 5, p(7) = 3, p(2) = 0.



- 与 8 相容的最大编号的任务是 5，p(8) = 5

定义 $OPT(j)$ 是有 j 个任务时的最优解，$OPT(0)=0$

我们的原问题为 $j=n$ 时的情况，考虑最优解发生的情形

- 若 OPT 选择任务 j，
    - 因为选择了任务 j，那么就不能选择 $p(j)+1, p(j)+2, \ldots, j-1$ 的所有任务
    - 它会在 $1,2,\ldots,p(j)$ 上做出最优的选择
    - $OPT(j)=OPT(p(j))+v_j$
- 若 OPT 没有选择任务 j
    - 它会在 $1,2,\ldots,j-1$ 上做出最优的选择
- OPT 将选择上述两种情况下更大的那一个

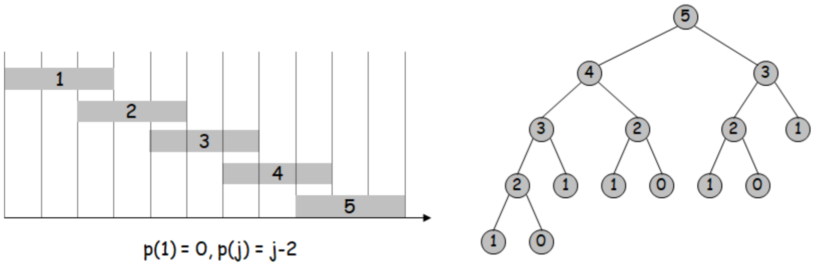
故我们给出以下表达形式

$$OPT(j)=\begin{cases}0 & \text{if j} = 0\\ \max\{v_j+OPT(p(j)), \mathrm{OPT}(j-1)\} & \text{otherwise}\end{cases}$$

```
输入: n, s1,...,sn, f1,...,fn, v1,...,vn;
按照结束时间排序，使得 f1 ≤ f2 ≤ ... ≤ fn;
计算 p(1), p(2),...,p(n);
Find-OPT(j):
    if(j == 0){
        return 0;
    }else{
        return max(v_j + Find-OPT(p(j)), Find-OPT(j-1));
    }
```

2.3 时间复杂度分析

如果我们真的按照所写的实现这个算法，那么最坏的情况下它将使用指数进行



- 如上图的示例中，为了计算 $OPT(5)$，必须递归计算 $OPT(4), OPT(3)$
- 继续调用浪费了大量的时间

事实上，我们并不是没有一个多项式时间的算法，观察到算法只是求解了 $n+1$ 个不同的子问题 $OPT(0), OPT(1), \ldots, OPT(n)$，我们可以在第一次计算的时候就把这些 OPT 值存在一个大家都可以访问的地方，然后后来的递归调用中只是使用这些预先算好的值，修改算法如下

```
输入: n, s1,...,sn, f1,...,fn, v1,...,vn;
按照完成时间排序，使得 f1 ≤ f2 ≤ ... ≤ fn;
计算 p(1), p(2),...,p(n);
Find-OPT(j):
    维护一个记录 OPT() 的数组 OPT;
    if(j == 0){
        return 0;
    }else if(OPT[j] is not null){
        return OPT[j];
    }else{
        OPT[j] = max(v_j + Find-OPT(p(j)), Find-OPT(j-1)));
        return OPT[j];
    }
```

该算法把时间复杂度优化到了 $O(n\ logn)$

动态规划算法给出了加权排序问题的最优值，如何得到最优解的具体的任务？

我们可以通过计算后存储在 `OPT[]` 中的值来获得，我们知道如何确定 j 是否属于最优解 $O(j)$，所以它可以通过 `OPT[]` 来找出最优解的区间集合

```
维护一个记录最优解集合的数组 S;
Find-Solution(j):
    if(j == 0){
        return S;
    }else{
        if(v_j + OPT[p(j)] ≥ OPT[j-1]){
            S.add(j);
            Find-Solution(p(j));
        }else{
            Find-Solution(j-1);
        }
    }
```

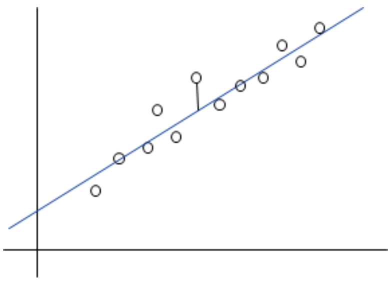
由于 `Find-Solution()` 只在确定更小的值上调用自己，因此它总的产生 $O(n)$ 次递归调用，并且每次调用只用常数时间

所以整个算法的时间复杂度为 $O(n)$

3. 分段最小二乘 Segmented Least Squares

3.1 问题描述

假设我们的数据由平面上的 n 个点的集合 P 组成，表示为 $(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)$，并且假设 $x_1<x_2<\ldots<x_n$，给定由方程 $y=ax+b$ 定义的直线 L，我们说 L 相对于 P 的误差是它对于 P 中的点的距离的平方和



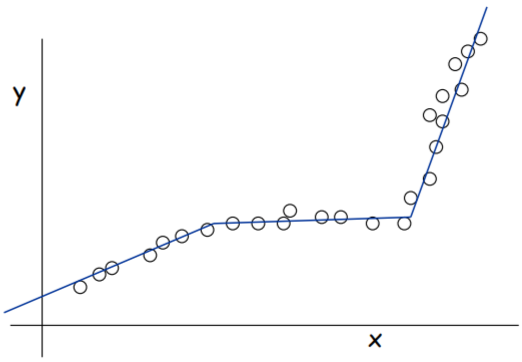
$$Error(L,P)=\sum_{i=1}^{n}(y_i-ax_i-b)^2$$

一个自然的目标是，找到具有最小误差的直线，当然经过数学计算，这条最小误差直线可以很容易地得出

$$a=\frac{n\sum_i x_iy_i-(\sum_i x_i)(\sum_i y_i)}{n\sum_i x_i^2-(\sum_i x_i)^2}$$

$$b=\frac{\sum_i y_i - a\sum_i x_i}{n}$$

现在我们有这样一类问题，有的点大致会位于多条直线上



我们提出一种新的问题：不是要找一条最佳拟合的直线，而是允许用任何最少一组直线穿过这些点，来很好的拟合这些点

给定一组点 $P=\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$，并且 $x_1<x_2<\ldots<x_n$，我们希望找到一系列直线来最小化某个优化目标 $f(x)$

- 考虑所有直线的误差和 E
- 考虑使用直线的数目 L

可以考虑某一个罚分 $E+cL$ ，使用某些常数 $c>0$

我们在分段最小二乘问题的目标是找到一个最小罚分的划分

3.2 算法

定义 $OPT(j)$ 是考虑了 n 个点 $p_1,p_2,\ldots,p_j$ 下的最小罚分，$OPT(0)=0$

定义 $e(i,j)$ 是对 $p_i,p_{i+1},\ldots,p_j$ 这些点用一条直线拟合所产生的最小二乘费用

假设最后一条直线包含了点 $p_i,p_{i+1},\ldots,p_j$，则最优解为

- $e(i,j)+c+OPT(i-1)$

由于我们不能确定 i 的位置在哪里，所有我们枚举所有可能，i 从 1 到 j，计算所有的罚分可能并取最小值，即满足下列表达式

$$OPT(j)=\begin{cases}0 & \text{if j}=0\\ \min_{1\le i\le j}\{e(i,j)+c+OPT(i-1)\} & \text{otherwise}\end{cases}$$

```
```

```
E[][]; // 记录最小二乘的数组
for (j = 1 to n){
    for(i = 1 to n){
        计算 e(i,j);
        E[i][j] = e(i,j);
    }
}

OPT[0] = 0;
for (j = 1 to n){
    OPT[j] = min_{1≤i≤j}{e(i,j)+c+OPT[i-1]};
}
return OPT[n]
```

3.3 时间复杂度分析

时间复杂度为 $O(n^3)$

4. 背包问题 Knapsack Problem

4.1 问题描述

有 n 个物品和一个背包

- 物品 i 有两个参数，权重 $w_i > 0$ 和收益 $v_i > 0$
- 背包有一个容量 W

目标：选择权重和不超过背包容积的问题，使得收益和最大

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

- 如上图，物品 3 和 4 的权重和为 11 = W，它们具有的总收益 40 最大

4.2 算法

定义 $OPT(i,w)$ 是从物品 $1,\dots,i$ 进行选择，并且当前背包的剩余容量为 w，$OPT(0,w)=0$

- 如果 OPT 不选择物品 i
    - OPT 会在物品 $1,2,\dots,i-1$，和容量为 w 下做出选择
    - $OPT(i,w)=OPT(i-1,w)$
- 如果 OPT 选择物品 i
    - OPT 会在物品 $1,2,\dots,i-1$，和容量为 $w-w_i$ 下做出选择
    - $OPT(i,w)=v_i+OPT(i-1,w-w_i)$

OPT 将选择上述两种情况下更大的那个，表达式如下

$$OPT(i,w)=\begin{cases}0 & \text{if } i=0\\ OPT(i-1,w) & \text{if } \mathrm{w_i} > \mathrm{w}\\ \max\{OPT(i-1,w),\quad v_i+OPT\,(i-1,w-w_i)\} & \text{otherwise}\end{cases}$$

```
for (w = 0 to W){
    OPT[0][w] =0
}
for (i = 1 to n){
    for(w = 1 to W){
        if(w_i > w){
            OPT[i][w] = OPT[i-1][w]
        }else{
            OPT[i][w] = Max{OPT[i-1][w], v_i + OPT[i-1][w-w_i]}
        }
    }
}
return OPT[n][W]
```

4.3 时间复杂度分析

算法的运行时间复杂度为 $O(nW)$

- 注意，它并不是关于输入规模的多项式算法
- W 是背包的容积
- 称为：伪多项式时间算法 Pseudo-polynomial

5. RNA 的次级结构 RNA Secondary Structure

5.1 问题描述

RNA 可以表达成一个由碱基构成的字符串 $B = b_1 b_2 \dots b_n$，碱基共有四种 $\{A, C, G, U\}$

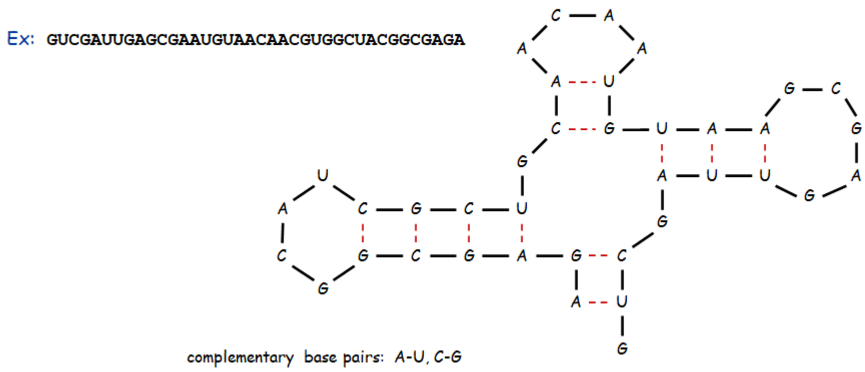
因为 RNA 是单链的，所以它倾向于回环和自己形成碱基对

次级结构

RNA 序列中碱基按照某种规则进行配对的情况

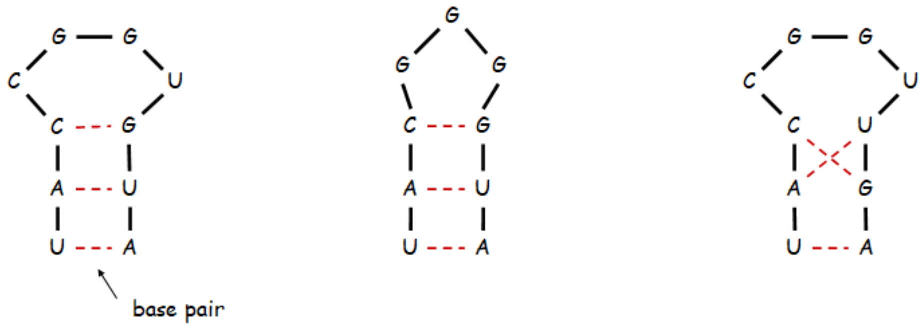
配对需要满足几个基本准则

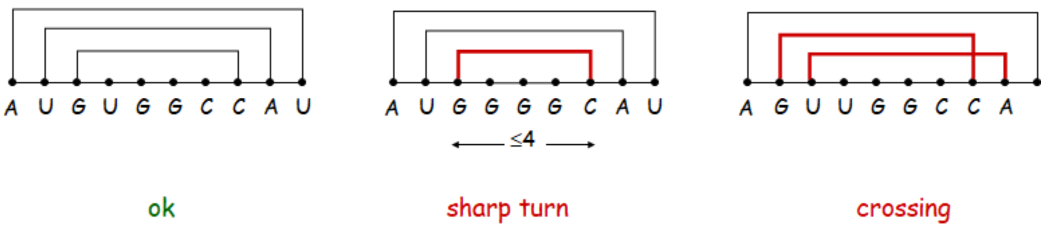
- 配对的情况只有 A-U、U-A、C-G、G-C 四种
- 没有急促的弯转，任何的配对 $(b_i, b_j) \in S$，它们之间至少间隔四个碱基，即 $i < j - 4$
- 配对不能有交叉，如果 $(b_i, b_j), (b_k, b_l)$ 是 S 中的两个配对，则不能有 $i < k < j < l$，只能要么是 $i < k < l < j$，要么是 $k < i < j < l$



- 上图所示的 RNA 形成了一个二级结构

目标：给定一个 RNA 分子 $B = b_1 b_2 \dots b_n$，找到二级结构中配对的集合 S，使得该集合的数目最大





- 第一个：合法
- 第二个：急促的弯转
- 第三个：配对交叉

5.2 算法

定义 $OPT(i,j)$ 是考虑序列 $b_ib_{i+1}\dots b_j$ 所对应的二级结构的最优值

- 如果 $i \ge j-4$
  - 因为是急促的弯转，不可能有配对，$OPT(i,j)=0$
- 如果 b_j 没有获得配对
  - $OPT(i,j)=OPT(i,j-1)$
- 如果 b_j 和某一个碱基 b_t 配对
  - 由碱基对不能交叉的限制，可以把问题分成子问题
  - $OPT(i,j)=1+max_t\{OPT(i,t-1)+OPT(t+1,j-1)\}$
  - 其中 i ≤ t < j-4，遍历所有的 t 的情况找到最大值

```
for(k = 5 to n-1){
    for(i = 1 to n-k){
        j = i + k;
        if(i >= j-4){
            OPT[i][j] = 0;
        }else{
            case1 = OPT[i][j-1];
            case2 = 0
            for(t = i to j-4){
```

```
                ti = 1 + max{OPT[i][t-1], OPT[t+1][j-1]};
                case2 = max{case2, ti};
            }
            OPT[i][j] = max{cas1, case2};
        }
    }
}
return OPT[1][n];
```

5.3 时间复杂度分析

时间复杂度为 $O(n^3)$

6. 序列比对 Sequence Alignment

6.1 概念定义

对于两个字符串，有多种对它们相似度的比对方式

错配 Mismatch

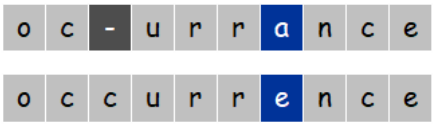
序列比对中的一个配对的两个字符不同

定义错配的惩罚参数 α_{pq}

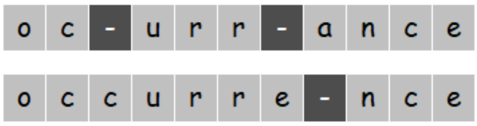
空隙 Gap

序列比对中一个配对的两个字符中，有一个是空的，用 `-` 表示

定义 Gap 的惩罚参数为 δ



1 mismatch, 1 gap



0 mismatches, 3 gaps

开销 Cost

Gap 和错配所产生的罚分总量

6.2 问题描述

给定两个字符串 $X = x_1x_2...x_m$ 和 $Y = y_1y_2...y_n$ 把它们进行比对，使得开销最小

序列比对满足如下性质

- 两边字符串里的每个字符只能组成某一个配对 / 发生 Gap
- 不能发生交叉，使得 $x_i - y_j$ 和 $x_k - y_l$ 配对，但是 $i < k, j > l$

那么两个字符串序列比对的总开销为

$$\text{cost}(M) = \underbrace{\sum_{(x_p y_j)\in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i:x_i \text{ unmatched}} \delta + \sum_{j:y_j \text{ unmatched}} \delta}_{\text{gap}}$$

6.3 算法

定义 $OPT(i, j)$ 是把 $x_1x_2...x_i$ 和 $y_1y_2...y_j$ 进行比对的最优目标值，
$OPT(i, 0) = i\delta, OPT(0, j) = j\delta$

- 如果 OPT 中 $x_i - y_j$ 配对
  - 付出错配可能罚分，以及继续比对 $x_1x_2...x_{i-1}$ 和 $y_1y_2...y_{j-1}$
  - $OPT(i, j) = \alpha_{pq} + OPT(i-1, j-1)$
- 如果 OPT 中 x_i 没有配对
  - 付出 Gap 的罚分，继续比对 $x_1x_2...x_{i-1}$ 和 $y_1y_2...y_j$
  - $OPT(i, j) = \delta + OPT(i, j-1)$

- 如果 OPT 中 y_j 没有配对
    - 付出 Gap 的罚分，继续比对 $x_1x_2...x_i$ 和 $y_1y_2...y_{j-1}$
    - $OPT(i,j) = \delta + OPT(i-1,j)$

OPT 将选择上面三种情况下最小的那一个，表达式如下

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_iy_j} + OPT(i-1,j-1) \\ \delta + OPT(i-1,j) \\ \delta + OPT(i,j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

```
for(i = 0 to m){
    OPT[i][0] = iδ;
}
for(j = 0 to m){
    OPT[0][j] = jδ;
}

for(i = 1 to m){
    for(j = 1 to n){
        case1 = α[xi][xj] + OPT[i-1][j-1];
        case2 = δ + OPT[i][j-1];
        case3 = δ + OPT[i-1][j]
        OPT[i][j] = min{case1,case2,case3}
    }
}

return OPT[m][n];
```

6.4 时间复杂度分析

算法的时间复杂度为 $O(mn)$

7. 最短路径 Shortest Paths

7.1 问题描述

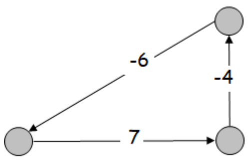
给定有向图 $G=(V,E)$，源点 s 和终点 t，每条边有一个可以为负的边权 c_{vw}

目标：找到一条从 s 到 t 的最短路径

7.2 概念定义

负圈 Negative cost cycle

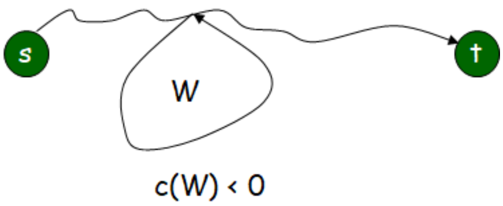
一个有向环，其中的边权之和为负



发现

如果从 s 到 t 会经过一个负圈，那么不会存在从 s 到 t 的最短路径

否则，则肯定存在一条路径



7.2 算法

由于边权可以为负，Dijkstra 算法失效，我们需要重新找到一种有效的算法

定义 $OPT(i,v)$ 是 v 到 t，最多用 i 条边最短路长度，令这条最短路径为 P，$OPT(0,v)=\infty$

- P 用了最多 $i-1$ 条边
    - $OPT(i,v)=OPT(i-1,v)$
- P 用了正好 i 条边

- 如果 (v, w) 是第一条边，那么 OPT 用了 (v, w) 边，然后选择 $w - t$ 的不超过 $i - 1$ 条边
    - $OPT(i, v) = c_{vw} + OPT(i - 1, w)$
- 由于 v 有多个出边，我们应该选择 v 的出边中返回开销最小的那一个

OPT 选择上面两种情况较小的那一个，表达式如下

$$OPT(i, v) = \begin{cases} 0 & \text{if i} = 0 \\ \min\{OPT(i-1, v), \quad \min_{(v,w)\in E}\{OPT(i-1, w) + c_{vw}\}\} & \text{otherwise} \end{cases}$$

```
for(node v ∈ V){
    M[0][v] = ∞
}

for(i = 1 to n - 1){
    for(node v ∈ V){
        M[i][v] = M[i-1][v]
    }
    for(edge (v,w) ∈ E){
        M[i][v] = min{M[i][v],M[i-1][w]+c_{vw}}
    }
}

return M[n-1][s]
```

7.3 时间复杂度分析

算法的时间复杂度为 $O(mn)$