第一遍看知识点，第二遍看证明，记住各种算法的复杂度

Stable matching

- Gale-Shapley algorithm **O(n2)**

Graph

- Adjacency matrix **Θ(n2) identify all edge; theta(1）check 1 edge; Space n2**
- Adjacency list **Space proportional to m + n. Checking if (u, v) is an edge takes O(deg(u)) time. Identifying all edges takes theta(m + n) time**
- BFS
- DFS
- Testing bipartiteness (use BFS) **若bfs同一层的节点存在边连着，则有odd length cycle。只能检测是否二分**
- Connectivity in Directed Graphs
    - Strong Connectivity **检测：正反图分别bfs，若都能达到所有点，则强连通** O(m+n)
- Topological order O(m+n)

Greedy

- Interval scheduling **权值为1，每次取earliest finish time 按finish time排序O(nlogn)，贪心O(n)**（Greedy algorithm stay ahead)
- Interval partitioning **排教室问题，按上课的start time排序，将教室按照当前课的finish time存在priority queue里。O(nlogn)** (Structral)
- Minimizing lateness **earliest deadline first, 按照ddl排序 O(nlogn)**（Exchange argument)
- Optimal caching
    - Farthest in the future （offline算法，事先知道所有访问)
    - LRU (k-compeititve)
    - LIFO (Arbitrarily bad)
- Shortest path in a graph (Dijkstra)
- MST (Kruskal, Prim, Reverse-delete)
    - Cut property **Cut为点集，cutset为cut向外连出的边集**
    - Cycle property
    - Cycle-cut intersection
    - Kruskal 用到了cycle property，用union-find **当m<=n2，即n比较大时，排序O(mlogn)（一般是O(mlogm))，并查集O(m α(m,n))**
    - Prim 用到了cut property，用binary heap O(mlogn)
    - Reverse-delete 类似反向kruskal，删最大边，且不使得不连通
- Clustering (Equivalent to finding an MST and deleting the k-1 most expensive edges.)
- Huffman
    - 使得ABL最优 Average bits per letter
    - 一个编码不能是另一个编码的前缀

- 优先对列排序f(字符频率)，O(nlogn)

Divide and Conquer

- 求解递推方程
  - 主定理
  - 递推树
  - 先猜想，后数学归纳证明
  - Telescoping
- Mergesort
  $T(n) = 2T(n/2) + O(n)$
  $T(n) = O(nlogn)$
- Counting inversions
  - Divide O(1)

    Conquer 2T(N/2)

    Count O(n)

    Merge O(n)
  - $T(n) \le T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) => T(n) = O(nlogn)$
- Closest pair of points
  - Divide: 划分点数各一半 O(nlogn)
  - Conquer: 两边找最近点 2T(n/2)
  - Combine:
    - 按y坐标排序 O(nlogn)
    - 移除所有离中线距离大于δ的点 O(n)
    - 对每个点找最近小于δ的点对 O(n) **最少只需考虑7个（或者11个，如果考虑3行正方形）**
  - 总体 $T(n) \le 2T(n/2) + O(nlogn) => T(n) = O(nlog^2 n)$
  - **优化：递归时类似mergesort，分别返回按y排序的点集（处理stripe中的点无需再次排序）和按x排序的点集（找stripe）。最终 O(nlogn)，combine时复杂度变为O(n)**
- Integer mutiplication

  If $T(n)$ obeys the following recurrence relation

  $$T(n) \le qT(n/2) + cn$$

  when n>2 and $T(2) \le c$.

  $T(\cdot)$ satisfying the above with $q > 2$ is bounded by $O(n^{\log_2 q})$.

  When q=3, $O(n^{\log_2 q}) = O(n^{1.585})$

  When q =4, $O(n^{\log_2 q}) = O(n^2)$

  - 竖式计算乘法 O(n2) bit operations
  - 简单分治Multiply four ½n-bit integers, recursively, add and shift **O(n2) bit operations**

- Karatsuba multiplication: Add two ½n bit integers. Multiply three ½n-bit integers, recursively. Add, subtract, and shift to obtain result. **O(n^1.585) bit operations** T(n) <= 3(T/2) + O(n)

- Convolution and FFT
  - Fourier theorem. [Fourier, Dirichlet, Riemann] Any periodic function can be expressed as the sum of a series of sinusoids.
  - Euler's identity
  - Polynomials: Coefficient representation/ Point-value representation
  - Conversion between two representation: C->P O(n^2) , P->C (n^3) (Both burte force) **P->C用快速矩阵乘法可达 O(n^2.376)**
  - FFT: C->P in O(nlogn)
  - Inverse FFT: P->C in O(nlogn)
  - Polynomial multiplication: O(nlogn), FFT C to P, multiplication in P only O(n), inverse FFT P to C.

DP (Memoization)

- Weighted interval scheduling (Binary choice) **brute force计算opt会达到指数级，需要用数组存**
  - 按finish time排序 O(nlogn)
  - 计算 p(.) O(nlogn)
  - 计算M[] O(n)
  - 输出选了哪些job O(n)

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

- Segmented Least Squares
  - O(n3).
  - Bottleneck = computing e(i, j) for O(n2) pairs, O(n) per pair using previous formula,
  - **DP部分O(n2)（j从1到n，每个j都需要O(n)来求min 这里体现了multiway)**
  - 预处理可达到总体O(n2)

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

```
The idea, whose details we will leave as an exercise for the reader, is to
first compute ei,j for all pairs (i, j) where j − i = 1, then for all pairs
where j − i = 2, then j − i = 3, and so forth. This way, when we get to a
particular ei,j value, we can use the ingredients of the calculation for
ei,j−1 to determine ei,j in constant time.
```

- Knapsack $\Theta(nW)$ Not polynomial in input size! "Pseudo-polynomial." (Adding a variable)

  先遍历n，再遍历w

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

- RNA Second structure (DP over intervals) **O(n3)**
  - No sharp turns (一组配对中间至少间隔4个点)
  - Watson-Crick
  - Non-crossing

(6.13) $\text{OPT}(i,j) = \max(\text{OPT}(i,j-1), \max(1+\text{OPT}(i,t-1)+\text{OPT}(t+1,j-1)))$, *where the max is taken over t such that b_t and b_j are an allowable base pair (under conditions (i) and (ii) from the definition of a secondary structure).*

  - 顺序：小区间到大区间 （j-i从小到大，体现到数组上是沿斜线，从左下到右上，一次填一个斜线）

RNA sequence *ACCGGUAGU*

	$j=6$	7	8	9
4	0	0	0	
3	0	0		
2	0			
$i=1$				

Initial values

	$j=6$	7	8	9
4	0	0	0	0
3	0	0	1	
2	0	0		
$i=1$	1			

Filling in the values for $k = 5$

	$j=6$	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	
$i=1$	1	1		

Filling in the values for $k = 6$

	$j=6$	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
$i=1$	1	1	1	

Filling in the values for $k = 7$

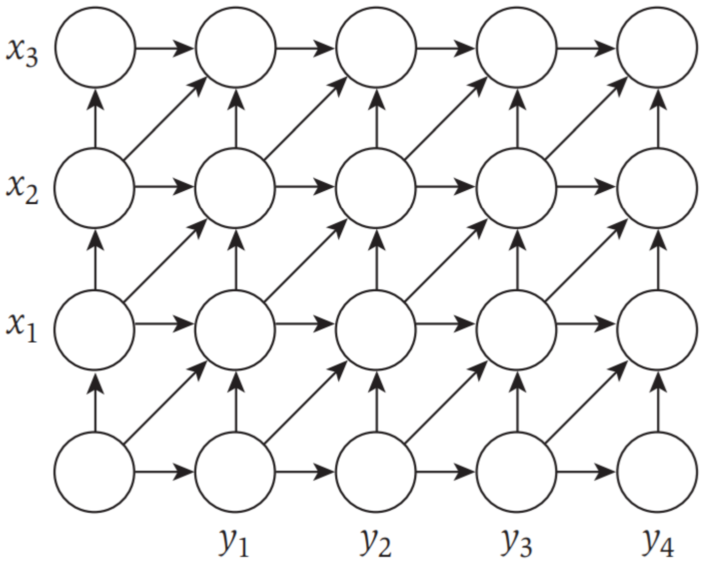
	$j=6$	7	8	9
4	0	0	0	0
3	0	0	1	1
2	0	0	1	1
$i=1$	1	1	1	2

Filling in the values for $k = 8$

  - 遍历：O(n2) 计算OPT: O(n)

- Sequence Allignment: $\Theta(mn)$ time and space

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




```
We number the rows from 0 to m and the columns from 0 to n; we denote
the node in the ith row and the jth column by the label (i, j). We put costs
on the edges of GXY: the cost of each horizontal and vertical edge is δ, and
the cost of the diagonal edge from (i − 1, j − 1) to (i, j) is αxiyj
Let f(i, j) denote the minimum cost of a path from (0, 0) to (i, j) in GXY.
Then for all i, j, we have f(i, j) = OPT(i, j).
```

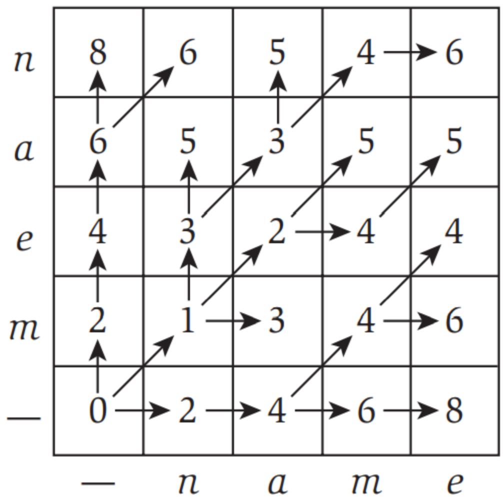


Figure 6.18 The OPT values for the problem of aligning the words *mean* to *name*.

- Sequence Allignment in linear space: $\Theta(m+n)$ space, $\Theta(mn)$ time
  - 用到了分治
  - **有时间再细看**
- Shortest path with negative edge
  - Negative cost cycle **存在负环则无最短路**
  - $$OPT(i,v) = \begin{cases} 0 & \text{if } i=0 \\ \min\left\{ OPT(i-1,\ v),\ \min\limits_{(v,w)\in E} \{ OPT(i-1,\ w)+c_{vw} \} \right\} & \text{otherwise} \end{cases}$$

OPT(n-1, v) = length of shortest v-t path

OPT(i,v)表示 **length of shortest v-t path P using at most i edge**

用这种方法空间复杂度为n2，时间mn

- Bellman-ford

```
Push-Based-Shortest-Path(G, s, t) {
   foreach node v ∈ V {
      M[v] ← ∞
      successor[v] ← ϕ
   }

   M[t] = 0
   for i = 1 to n-1 {
      foreach node w ∈ V {
      if (M[w] has been updated in previous iteration) {
         foreach node v such that (v, w) ∈ E {
            if (M[v] > M[w] + c_vw) {
               M[v] ← M[w] + c_vw
               successor[v] ← w
            }
         }
      }
      If no M[w] value changed in iteration i, stop.
   }
}
```

空间降到O(m+n)，时间复杂度O(mn)，但实际中会快很多。最多遍历点数-1次（路径最多n-1条边），上一次没更新dis的话则提前终止

- **Distance vector protocol (不知道是啥)**

Network flow

- **s-t cut** : capacity of a cut
- **s-t flow**: value of a flow
- Flow-value lemma

```
Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent
across the cut is equal to the amount leaving s
```

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e) = v(f)$$

- weak duality lemma
- Integrality theorem
- Ford-Fulkerson O(Cmn) **找一个augment path，用bfs O(m+n)，更新用 O(n)，因为最长路径为O(n-1)**
- Capacity Scaling O(m2logc): O(mlogc) augmentation, finding each O(m) **Find large bottleneck capacity**
- BFS **Find shortest augmenting path**

- Bipartite Matching O(mn) c=1 **虚拟s和t,s到其他点的cap是1，二分图之间的cap是无穷，二分图到t的cap是1**
- 霍尔婚姻定理
- Disjoint Path in Directed graph （数量一定，路线不一定唯一）
    - Network connectivity (Menger's theorem) O(mn) c=1
- Disjoint Path in Undirected graph O(mn) c=1 **把每条边都看成两条有向边，且必须满足f(e)=0 or f(e')=0 or both 0，否则会出现同一条边走两次** （Menger's theorem still applies)
- Extensions
    - Multiple sources/sinks **建一个大s，连到sources上，capacity都为无穷；建一个大t，连到sinks上，capacity都为无穷**
    - Circulation with supplies & demands

```
Add new source s and sink t.
For each v with d(v) < 0, add edge (s, v) with capacity -d(v).
For each v with d(v) > 0, add edge (v, t) with capacity d(v)
```