导论

性质：
有穷性、确定性、可行性

插入排序伪代码：

```
for j=2 to n do
  key = A[j]
  i = j - 1
  while i > 0 and A[i] > key do
    A[i + 1] = A[i]
    i = i - 1
  A[i + 1] = key
```

$T = O(n^2)$
$S = O(1)$

归并排序伪代码：

```
mergeSort(A, p, r)
  if p < r then
    q = (p + r) / 2
    mergeSort(A, p, q)
    mergeSort(A, q + 1, r)
    merge(A, p, q, r)

merge(A, p, q, r)
  n1 = q - p + 1
  n2 = r - q
  L[1,..,n1 + 1] = A[p..q]
  R[1,..,n2 + 1] = A[q + 1..r]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = j = 1
  for k = p to r do
    if L[i] ≤ R[j] then
      A[k] = L[i]
      i = i + 1
    else
      A[k] = R[j]
      j = j + 1
```

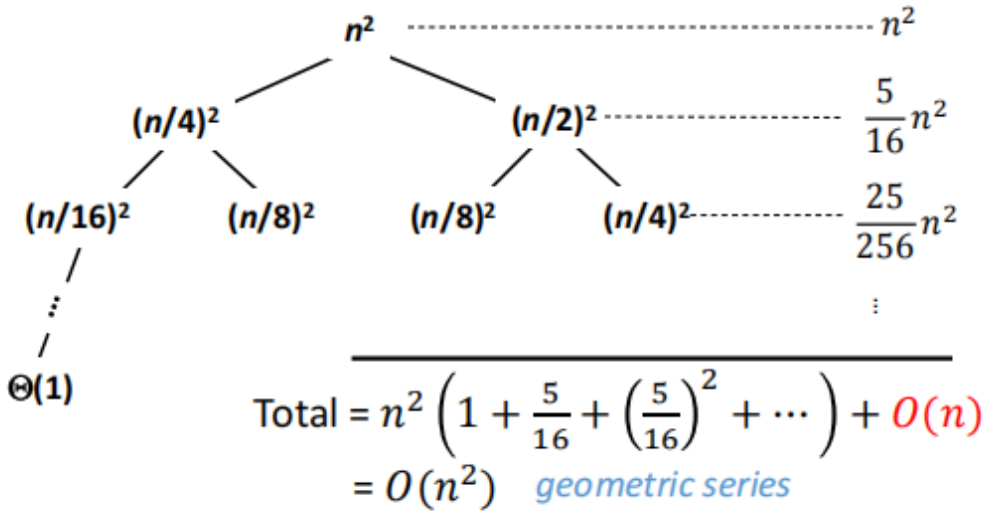
$T = O(n * \log n)$
$S = O(n)$

分析：
$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = c > 0 \Rightarrow \frac{1}{2}\cdot c\cdot g(n) \le f(n) \le 2\cdot c\cdot g(n) \Rightarrow f(n) = \theta(g(n))$
若 c=0,则$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0 \Rightarrow f(n) = o(g(n))$ but not $\theta(g(n))$

主方法：
$T(n) = a\cdot T\left(\frac{n}{b}\right) + f(n)$
$n^{\log_b a} > f(n), T(n) = \theta\left(n^{\log_b a}\right)$
$n^{\log_b a} = f(n), T(n) = \theta(f(n)\cdot\log(n))$
$n^{\log_b a} < f(n), T(n) = \theta(f(n))$
tips:
a 为常数,子问题数≥1,f(n)系数为正,不可波动$n^{\log_b a}$在定义域内与 f(n)大小关系确定

递归树方法:**When you say nothing at all**

1. 采用Recursion-Tree方法求解递推式
$T(n) = T(n/4) + T(n/2) + n^2.$



最后的 O(n)是处理每一层的工作量

分治算法

最大子数组伪代码:

```
maxSubArray(A, low, high)
  if low == high then
    return A[low]
  mid = (low + high) / 2
  leftSum = maxSubArray(A, low, mid)
  rightSum = maxSubArray(A, mid + 1, high)
  crossSum = maxCrossingSum(A, low, mid, high)
  return max(leftSum, rightSum, crossSum)

maxCrossingSum(A, low, mid, high)
  leftSum = -∞
  sum = 0
  for i = mid downto low do
    sum = sum + A[i]
    if sum > leftSum then
      leftSum = sum
  rightSum = -∞
  sum = 0
  for j = mid + 1 to high do
    sum = sum + A[j]
    if sum > rightSum then
```

```
      rightSum = sum
  return leftSum + rightSum
```

$T(n) = 2T(\frac{n}{2}) + O(n)$
$T(n) = O(n \log n)$

寻找第 **k** 小元素伪代码:

```
findKth(A, k)
  if len(A) == 1 then
    return A[0]
  pivot = random(A)
  low = []
  high = []
  for x in A do
    if x < pivot then
      low.append(x)
    else if x > pivot then
      high.append(x)
  if k <= len(low) then
    return findKth(low, k)
  else if k > len(A) - len(high) then
    return findKth(high, k - (len(A) - len(high)))
  else
    return pivot
```

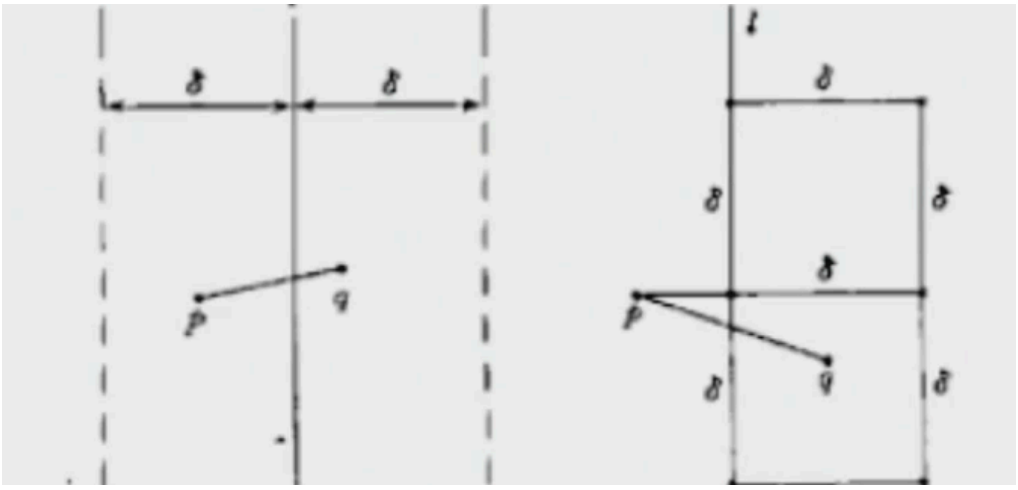
$T(n) = T(\frac{n}{2}) + O(n)$
$T(n) = O(n)$

最近点对问题:

解决:
分成S_l和S_r两部分,就可以分别得到S_l与S_r中的最近距离δ_l与δ_r,但集合 S 的最近点对还可能分别属于两边的点组成的δ_m,需要合并$\delta_* = \min\{\delta_l, \delta_r, \delta_m\}$

计算δ_p
为此我们设$\delta_p = \min\{\delta_l, \delta_r\}$,并在分开子问题的直线 L 左右两边分别画出一条宽度为δ的平行线,在这两条线之间的点对才有可能是最近点对,因此我们只需要考虑这些点对即可.

计算δ_m



如图所示,若我们考察 p 点,则 p 点若能更新最小值,一定满足他的 partner 为常数个且落在右侧 $\delta \times 2 \cdot \delta$的区域内,不妨将这些点按 y 坐标从小到大排序,设为$P_1, P_2, ..., P_m$,则P_i的 partner 只能是 $P_{\{i+1\}}, P_{\{i+2\}}, ..., P_{\{i+7\}}$中的一个(p 点前后各四个),因此只需要检查 8 个点即可(因为如果将右侧划分为六个$\frac{1}{2}\delta \times \frac{2}{3}\delta$的六个等大区域,一个区域内最多存在一个点,若有两个点则距离一定小于δ_p,与初始条件矛盾)

伪代码:

```
closestPair(P)
```

$\delta_m \leftarrow \infty$

```
if len(P) <= 3 then
  return bruteForce($delta_*$)
else
  mid = len(P) / 2
  left = P[:mid]
  right = P[mid:]
  delta_l = closestPair(left)
  delta_r = closestPair(right)
  delta_p = min(delta_l, delta_r)
  delta_m = closestSplitPair(P, delta_p)
  return min(delta_p, delta_m)
closestSplitPair(P, delta_p)
  min_dist = delta_p
```

$S_l' \leftarrow S_l$中介于$[x_m - \delta_p, x_m]$的点
$S_r' \leftarrow S_r$中介于$[x_m, x_m + \delta_p]$的点
S_p ←从 y 坐标从小到大提取$S_l' \leftarrow S_l$和$S_r' \leftarrow S_r$中的点

```
  for i in range(len(strip)) do
```

if $S_{p[i]} \in S_l'$then

```
  for j in range(max(i-4, 0), min(i + 4, len(strip)-1)) do
    dist = distance(strip[i], strip[j])
    if dist < min_dist & i!=j then
      min_dist = dist
return min_dist
```

逆序对计算伪代码:

```
countInversions(A)
  if len(A) <= 1 then
    return 0
  mid = len(A) / 2
  left = A[:mid]
  right = A[mid:]
  count = countInversions(left) + countInversions(right)
  i = j = k = 0
  while i < len(left) and j < len(right) do
    if left[i] <= right[j] then
      A[k] = left[i]
      i += 1
    else
      A[k] = right[j]
      count += len(left) - i #这里用到了线代第一章的def
      j += 1
```

```
    k += 1
  while i < len(left) do
    A[k] = left[i]
    i += 1; k += 1;
  while j < len(right) do
    A[k] = right[j]
    j += 1; k += 1;
```

Strassen 矩阵乘法分析:

$$A \times B = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \\ B_{31} & B_{32} & B_{33} & B_{34} \\ B_{41} & B_{42} & B_{43} & B_{44} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \\ C_{41} & C_{42} & C_{43} & C_{44} \end{bmatrix}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} + \begin{bmatrix} A_{13} & A_{14} \\ A_{23} & A_{24} \end{bmatrix} \begin{bmatrix} B_{31} & B_{32} \\ B_{41} & B_{42} \end{bmatrix}$$

$$\begin{bmatrix} C_{33} & C_{34} \\ C_{43} & C_{44} \end{bmatrix} = \begin{bmatrix} A_{31} & A_{32} \\ A_{41} & A_{42} \end{bmatrix} \begin{bmatrix} B_{13} & B_{14} \\ B_{23} & B_{24} \end{bmatrix} + \begin{bmatrix} A_{33} & A_{34} \\ A_{43} & A_{44} \end{bmatrix} \begin{bmatrix} B_{33} & B_{34} \\ B_{43} & B_{44} \end{bmatrix}$$

Figure 1: 优化前

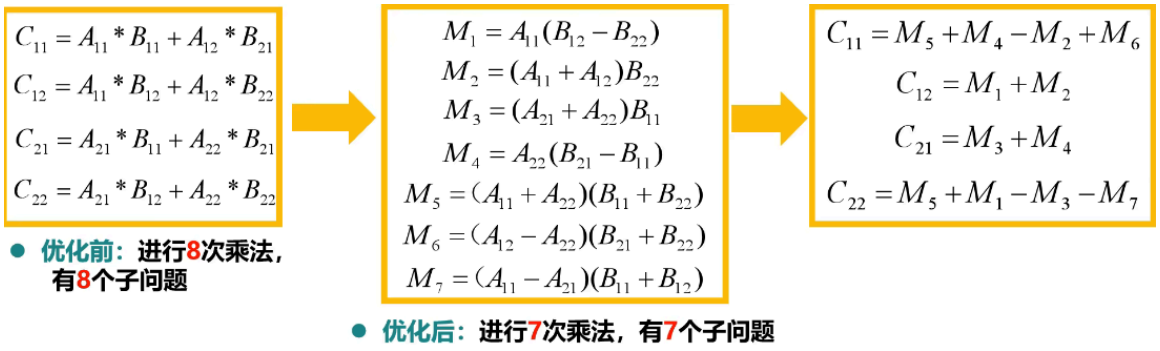


Figure 2: 优化后

```
Strassen (n, A, B)
if (n = 1) return A × B.
P1 ← Strassen (n/2, A11, B12 - B22)
P2 ← Strassen (n/2, A11 + A12, B22)
P3 ← Strassen (n/2, A21 + A22, B11)
P4 ← Strassen (n/2, A22, B21 - B11)
P5 ← Strassen (n/2, A11 + A22, B11 + B22)
P6 ← Strassen (n/2, A12 - A22, B21 + B22)
P7 ← Strassen (n/2, A11 - A21, B11 + B12)
C_11= P5 + P4 - P2 + P6
C_12= P1 + P2
C_21= P3 + P4
C_22= P1 + P5 - P3 - P7
return C
```

贪心算法

Dijkstra 伪代码:

```
dijkstra(G, s)
  dist[s] = 0
  for each vertex v in G do
    if v != s then
      dist[v] = ∞
    prev[v] = null
  Q = all vertices in G
  while Q is not empty do
    u = vertex in Q with min dist[u]
    remove u from Q
    for each neighbor v of u do
      alt = dist[u] + length(u, v)
      if alt < dist[v] then
        dist[v] = alt
        prev[v] = u
```

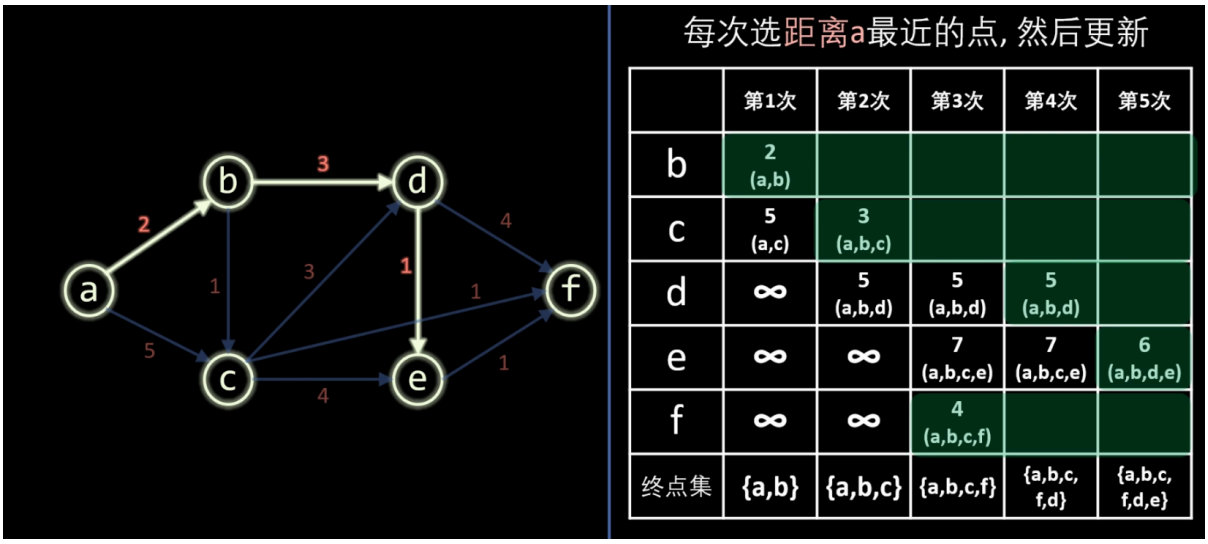


Figure 3: 老朋友了

$$T = O((V + E)\log(V))$$
$$S = O(V)$$

Prim 伪代码(一种加点的算法):

```
prim(G, start)
  for each vertex v in G do
    key[v] = ∞
    parent[v] = null
  key[start] = 0
  Q = G[0...n-1]
  while Q is not empty do
    u = vertex in Q with min key[u]
    remove u from Q
    for each neighbor v of u do
      if v in Q and length(u, v) < key[v] then
        parent[v] = u
        key[v] = length(u, v)
  return parent
```

$T = O((V + E) \log V)$
$S = O(V)$

Kruskal 伪代码(并查集实现,一种加边的算法):

```
kruskal(G)
  edges = all edges in G sorted by weight
  parent = [] # Union-Find structure
  for each vertex v in G do
    parent[v] = v
  mst = []
  for each edge (u, v) in edges do
    if find(parent, u) != find(parent, v) then
      union(parent, u, v)
      mst.append((u, v))
  return mst
```

$T = O(E \log E)$
$S = O(V)$

LP 问题

可行域:

图解法,可 LP 求解的问题目标函数与约束条件都是仿射的,二维的一定可图解
不难发现,LP 的可行解在可行域上的表现形式一定为某个 vertex,因此只需要找到可行域的顶点即可

单纯形法:

不难发现,当问题拓展为一般 LP 问题时,若引入松弛变量使之维度达到更高,单纯形法的每一步都是沿着可行域的边走到下一个顶点,因此每一步都可能找到一个更优的解,直到到达最优顶点
Q:
$\max z = 50x_1 + 40x_2$ s.t.
$3x_1 + 5x_2 + x_3 = 150$
$x_2 + x_4 = 20$
$8x_1 + 5x_2 + x_5 = 300$
$x_i \geq 0$ A:
这里引入了松弛变量x_3, x_4, x_5使得原问题变为等式约束,此时他们一定能构成一个单位矩阵(基变量因此得名),此后我们在寻找x_c的时候每次都选择检验数最大的x_i作为x_c(最后要两个都小于等于 0,每次都要降低更快),然后将新的入基,将比值最低的出基(前面的 intuition)通过标准行列变换得到新的单位基向量
设 x_i 的系数为 c_i,当前所选基变量 x_j 的系数为 c_j,$\delta_i = c_i - \sum_{i=1}^{n-j} c_{i,j} \cdot c_j$

		50	40	0	0	0		
		x_1	x_2	x_3	x_4	x_5	B_i	$\frac{B}{x_{cj}}$
0	x_3	3	5	1	0	0	150	150/3
0	x_4	0	1	0	1	0	20	∞
0	x_5	8	5	0	0	1	300	300/8
	δ_i	50	40	0	0	0		

此时$x_c = x_1$,x_5出基,

		50	40	0	0	0		
		x_1	x_2	x_3	x_4	x_5	B_i	$\frac{B}{x_c}$
0	x_3	0	25/8	1	0	$-\frac{3}{8}$	$\frac{75}{2}$	12
0	x_4	0	1	0	1	0	20	∞
50	x_1	1	$\frac{5}{8}$	0	0	$\frac{1}{8}$	$\frac{75}{2}$	60
	δ_i	0	$\frac{35}{4}$	0	0	$-\frac{25}{4}$		

不难发现此时通过初等变换得到了单位矩阵,同时B_i的值也随之改变,下略
最终当x_1, x_2都入基完成,检验数已经小于等于 0,因此到达最优解

动态规划

加权区间调度问题伪代码:

```
sort intervals by finish time\
dp[i] = 0 #dp[i]表示只考虑前i个能获得的最大权重\
compute p[i] for each interval i #意味着与作业i不冲突且结束时间最早的前序作业索引
for i = 1 to n do
  dp[i] = max(dp[i-1], dp[p[i]] + weight[i])
return dp[n]
```

最小二乘拟合伪代码:

```
leastSquaresFit(points,&cost_estimate)
for j=1 to n
  for i=1 to j do
compute e(i,j)
dp[0]=0
for i=1 to n
  for j=1 to i-1 do
```

$dp[i] \leftarrow \min_{1\le j\le i}\{e(i,j) + \text{cost\_estimate} + dp[j-1]\}$

```
return dp[n]
```

背包问题分析:
0-1 背包问题的状态转移方程为:

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

其中w_i为第 i 个物品的重量, v_i为第 i 个物品的价值, w为背包容量, i为物品索引,dp[i]表示选择到第 i 个物品
同理不难知道一般情况下完全背包问题的状态转移方程为:

$$\text{dp}[i][j] = \max(\text{dp}[i-1][j], \text{dp}[i][j-v[i]] + w[i])$$

此时的 i 表示的是种类, j 表示的是背包容量, dp[i][j]表示选择到第 i 种物品时的最大价值,在进行滚动数组的时候都是自底向上的

RNA 的二级结构预测:
tips:RNA 需要间隔至少四个单碱基才能进行二级结构的折叠

伪代码:

```
for k=5 to n-1 do
  for i=1 to n-k do
    j=i+k
```

for each b_t paired with b_j $(i \leq t < j-4)$

```
      T=1+dp[i][t-1]+dp[t+1][j-1]
dp[i,j]=max(dp[i,j-1],T)
return dp[1][n]
```

tips:此时画的矩阵为上三角矩阵,左下角为(1,6),刚好相差 5,且填数值时按对角线往右下顺序

弗洛伊德算法:

若图不是很复杂时使用

想象有一个图$0 \xrightarrow{6} 1, 0 \xleftarrow{10} 1, 1 \xrightarrow{4} 2, 0 \xrightarrow{13} 2, 0 \xleftarrow{5} 2,$

初始化两个二维矩阵D_i和P_i,其中D_i表示从 i 点到 j 点的最短路径长度,若不存在则为∞,若存在则为边权重,而P_i表示从 i 点到 j 点的前驱节点,若不存在则为-1

0	6	13
10	0	4
5	∞	0

Table 1: 初始距离矩阵 D^{-1}

−1	0	0
1	−1	−1
2	−1	−1

Table 2: 初始前驱节点矩阵 P^{-1}

0	6	13
10	0	4
5	11	0

Table 3: D^0表示中继节点为 0 的情况

−1	0	0
1	−1	1
2	0	−1

Table 4: 改变后的前驱节点矩阵 P^0

以此类推,直到所有节点都作为中继节点遍历完毕,最终得到的D^n即为从 i 点到 j 点的最短路径长度,而P^n则为从 i 点到 j 点的前驱节点,若不存在则为-1

伪代码:

```
floyd(G)
  for k = 1 to n do
    for i = 1 to n do
      for j = 1 to n do
        if dist[i][j] > dist[i][k] + dist[k][j] then
          dist[i][j] = dist[i][k] + dist[k][j]
  return dist
```

编辑距离问题:

tips:编辑距离问题是指将一个字符串转换为另一个字符串所需的最小操作数,操作包括插入、删除和替换

假设现在有一个字符串长度 m,一个字符串长度 n,定义 dp[i][j]为将字符串 s1 的前 i 个字符转换为 s2 的前 j 个字符所需的最小操作数,则有:

在 word1 中插入字符等价于在 word2 中删除字符,因此可以将问题转化为在 word1 中删除字符,在 word2 中插入字符,或者替换字符

$$\mathrm{dp}(i,j)=\begin{cases}0 & \text{if i=0 and j=0}\\ i & \text{if j=0}\\ j & \text{if i=0}\\ \mathrm{dp}(i-1,j-1) & \text{if s1[i - 1]=s2[j - 1]}\\ \min\{\mathrm{dp}(i-1,j),\ \mathrm{dp}(i,j-1),\ \mathrm{dp}(i-1,j-1)\}+1 & \text{otherwise}\end{cases}$$

LCS 问题:

最长公共子序列:

$$\mathrm{dp}(i,j)=\begin{cases}0 & \text{if i=0 or j=0}\\ \mathrm{dp}(i-1,j-1)+1 & \text{if s1[i - 1]=s2[j - 1]}\\ \max\{\mathrm{dp}(i-1,j),\ \mathrm{dp}(i,j-1)\} & \text{if s1[i - 1]!=s2[j - 1]}\end{cases}$$

最长公共子串:

$$\mathrm{dp}(i,j)=\begin{cases}0 & \text{if i=0 or j=0}\\ \mathrm{dp}(i-1,j-1)+1 & \text{if s1[i - 1]=s2[j - 1]}\\ 0 & \text{if s1[i - 1]!=s2[j - 1]}\end{cases}$$

链式矩阵乘法:

给定一系列矩阵$A_1, A_2, ..., A_n$,其有r_i行,c_i列,定义 dp[i][j]为从第 i 个矩阵到第 j 个矩阵的最小乘法次数,则有:

$$\mathrm{dp}(i,j)=\begin{cases}0 & \text{if i=j}\\ \min_{i\le k<j}\{\mathrm{dp}(i,k)+\mathrm{dp}(k+1,j)+r_i\cdot c_k\cdot c_j\} & \text{otherwise}\end{cases}$$

画图时是一个下三角矩阵,按对角线向右上角填数值

多源最短路径:

tips:显然的是,当路径中存在负权环的时候,无最短路径,这个时候 dijk 和 floyd 死循环

定义 dp[i][v]为从 v 点到指定目标点,设为 t 的最短路径长度,则有:

$$\mathrm{dp}(i,v)=\begin{cases}\infty & \text{if i = 0}\\ \min\Big\{\mathrm{dp}(i-1,v),\min_{((v,w))\in E}\{\mathrm{dp}(i-1,w)+c(v,w)\}\Big\} & \text{otherwise}\end{cases}$$

伪代码:

```
for each node v in V do
  dp[0][v] = ∞
dp[0][t] = 0
for i=1 to n do
  for each node v in V do
    dp[i][v] = dp[i-1][v]
    for each edge (v, w) in E do
      if dp[i][w] + c(v, w) < dp[i+1][v] then
        dp[i+1][v] = dp[i][w] + c(v, w)
```

回溯(分支限界就是优化的一般回溯):

说白了就是根据“选或不选”画一棵树

N 皇后问题代码:

```
void put_queen(int x,int y,vector<vector<int>>&attack)
{
  static int dx[8]={-1,-1,-1,0,0,1,1,1};
  static int dy[8]={-1,0,1,-1,1,-1,0,1};
  attack[x][y]=1;

for(int i=1;i<attack.size();i++{
  for(int j=0;j<8;J++)
  {
    int nx=x+dx[j];
    int ny=y+dy[j];
  }
  if(nx>=0&&nx<attack.size()&&ny>=0&&ny<attack.size())
    attack[nx][ny]=1;
  }
}
void backtrack(int k,int n,vector<string>&queen,
vector<vector<int>>&attack,vector<vector<string>>&solutions)
{
  if(k==n)
  {
    solutions.push_back(queen);
    return;
  }
  for(int i=0;i<n;i++)
  {
    if(attack[k][i]==0)
    {
      queen[k][i]='Q';
      vector<vector<int>>temp=attack;
      put_queen(k,i,temp);
      backtrack(k+1,n,queen,temp,solutions);
      queen[k][i]='.';
    }
  }
}
vector<vector<string>> solveNQueens(int n) {
  vector<string> queen;
  vector<vector<int>> attack;
  vector<vector<string>> solutions;
  for(int i=0;i<n;i++{
    attack.push_back(vector<int>());
    for(int j=0;j<n;j++){
      attack[i].push_back(0);
    }
    queen.push_back("");
    queen[i].append(n, '.');
  }
  backtrack(0, n, queen, attack, solutions);
  return solutions;
}
```

装船问题:
先将第一艘尽可能地装满,等价于选取全体集装箱的一个子集,使得该自己的重量之和最接近c_1,再看剩下的总重量与c_2的大小关系
分支限界剪枝：计算当前路径剩余货物全选为上界,按上界从大到小扩展节点(优先队列),丢弃上界 $\leq$ 当前最优解的分支

着色问题:

```
def graph_coloring_all_solutions(G, m):
    n = len(G.V)                    # 顶点数量
    colors = [0] * n                # 解向量 (x1, x2, ..., xn)
    solutions = []                  # 存储所有合法解
    adjacent_list = G.adj           # 邻接表表示图

    def is_valid(k, color):
        for neighbor in adjacent_list[k]:
            if colors[neighbor] == color:
                return False
        return True

    def backtrack(k):
        if k == n:                  # 所有顶点已着色，找到可行解
            solutions.append(colors.copy())
            return

        for color in range(1, m + 1):
            if is_valid(k, color):          # 约束条件：颜色不冲突
                colors[k] = color           # 选择颜色
                backtrack(k + 1)            # 递归处理下一个顶点
                colors[k] = 0               # 回溯，撤销选择

    backtrack(0)
    return solutions
```

优化方法:
先着色度数高的点,尽早触发剪枝

完全无向图的旅行商问题:
选取途中开销最小的一条边 w,则 n*w(n 为哈密顿回路边的条数)必然是一个下界
在完全图 G 中,对每个顶点v_i,$w_{i,1}$,$w_{i,2}$分别表示连接v_i的两条开销最小的边，令$w_i = w_{i,1} + w_{i,2}$,$\sum_i w_i \leq 2 \cdot \mathrm{Dist}_{\mathrm{tsp}}$,显然$\frac{1}{2} \cdot \sum_i w_i$也是一个下界
pf:
设旅行商回路一次经过节点 1 n,开销为$p_1 \sim p_n$,显然对于$v_i : p_{i-1} + p_i \geq w_i$(1 的上一位是 n),两边相加得证，$\left\lceil 2 \cdot \sum_i w_i \right\rceil$可选为一个下限界
对于上界直接使用贪心算法,每次均选择顺次时该最短的,剪枝是把这个上限界小于下限界的分支删去

蒙特卡洛方法:
多次模拟实验计算所有模拟实验结果的均值，可以近似地得到目标值的期望(如估算平均需要多少次节点计算才能得到回溯问题的最优解)