算法理解题

#1 数字三角形

- 给定一个数字三角形，从顶至底有多条路径，每一步可沿左斜线向下或沿右斜线向下，路径所经过的数字之和为路径得分，请求出最小路径得分和相应路径。
- 动态规划

```
for(int i = n - 1; i >= 1; i++)
    for(int j = 1; j <= i; j++)
        dp[i][j] = min(dp[i + 1][j],dp[i + 1][j + 1]) + a[i][j];
```

#2 背包问题

- 部分背包问题
  - 物品可分
    - 首先计算出每个物体的单位价值（v/w），然后从大到小排序。按照排好的顺序选择物品放入背包，由于物品可以部分装入，所以装完后背包的重量一定为c

```
// a[] 各个物品
// ans 最大价值
// c 当前容量
sort(a);
c = W;
for( i = 1; i <= n; i++){
    if(c < w[i]) break;
    c -= w[i];
    ans += v[i];
}
if(i <= n) ans += (c/w[i]) * v[i];
```

- 01背包
  - 物品不可分，只能选一次
  - 动态规划

```
// f[i] 表示容量为i时最大价值
for(int i = 1;i <= n; i++){
    for(int j = W; j >= w[i]; j--)
        f[j] = max(f[j],f[j - w[i]] + v[i]);
}
return f[V];
```

- 完全背包
  - 物品不可分，每个物品无数个
  - dp

```
// f[i] 表示容量为i时最大价值
for(int i = 1;i <= n; i++){
    for(int j = w[j]; j <= W; j++)
        f[j] = max(f[j],f[j - w[i]] + v[i]);
}
return f[V];
```

#3婚姻稳定问题

- 当前有N位男生和N位女生最后要组成稳定的婚姻家庭，过程开始之前男生和女生在各自的心目中都按照喜爱程度对N位异性有了各自的排序，男生和女生结婚后，对于每一对男生女生，不会出现比起当前匹配的伴侣互相更喜爱的一对男生女生，即可认为婚姻是稳定的。
- 做法：
  - 先对所有男士进行落选标记，称其为自由男。当存在自由男时，进行以下操作：
  - 每一位自由男在所有尚未拒绝她的女士中选择一位被他排名最优先的女士；
  - 每一位女士将正在追求她的自由男与其当前男友进行比较，选择其中排名优先的男士作为其男友，即若自由男优于当前男友，则抛弃前男友；否则保留其男友，拒绝自由男。
  - 若某男士被其女友抛弃，重新变成自由男。

#4哈夫曼编码

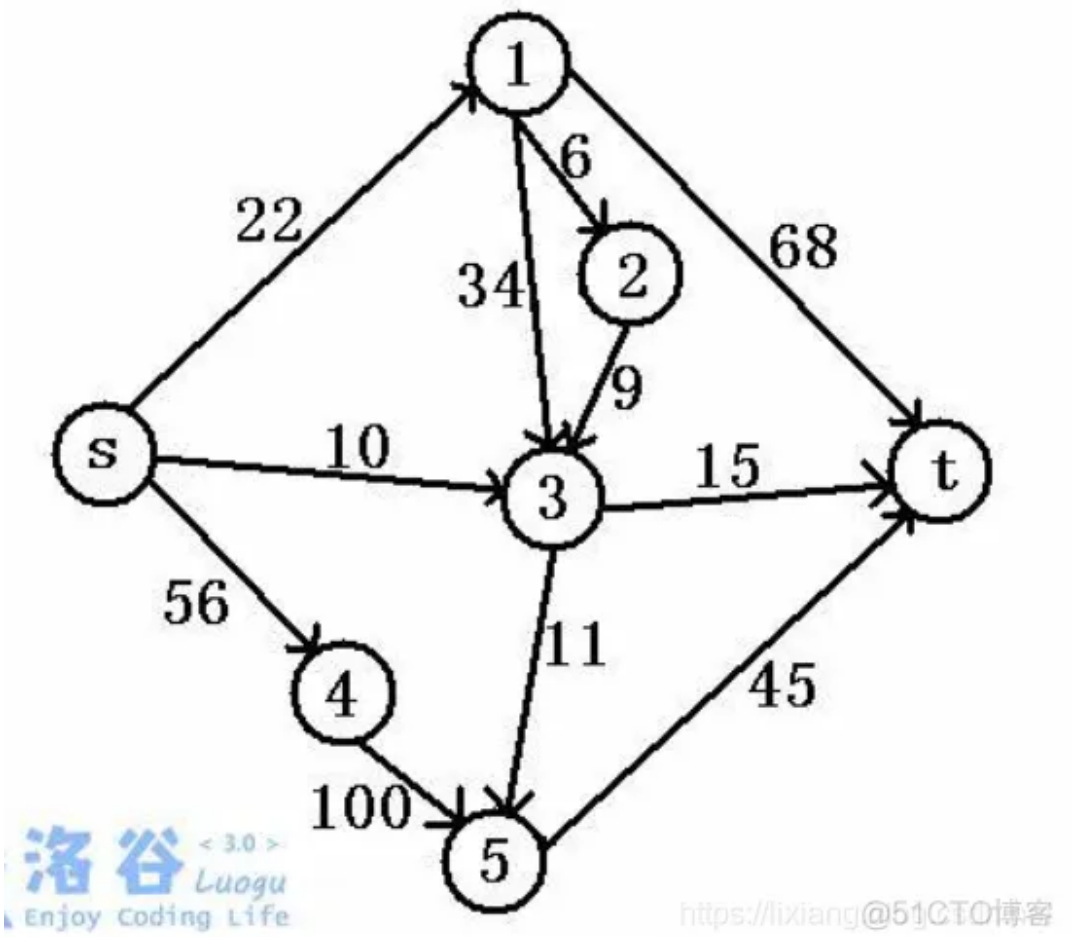
- 假设有一个字符串 "abracadabra"，我们来求解它的哈夫曼编码。
- 统计字符频率
  - 遍历字符串，统计每个字符出现的次数，得到以下频率表：

    ```
    a: 5
    b: 2
    r: 2
    c: 1
    d: 1
    ```

  - 构建哈夫曼树：
    - 创建叶子节点：根据字符频率创建与字符对应的叶子节点。
    - 构建哈夫曼树：重复以下步骤直到只剩下一个节点：
      - 从频率表中选择两个频率最低的节点。
      - 创建一个新的父节点，频率为这两个节点的频率之和。
      - 将选中的两个节点作为新节点的左右子节点。
      - 将新节点加入频率表。
  - 分配编码：
    - 从根节点出发，沿着左子树路径标记为0，沿着右子树路径标记为1。
    - 对于每个叶子节点，记录从根节点到该叶子节点的路径表示的编码。

#5给定网络G，求最大流、最小割、求最短路

- 最大流
  - 简单来说就是水流从一个源点s通过很多路径，经过很多点，到达汇点t，问你最多能有多少水能够到达t点。



- 从s到t经过若干个点，若干条边，每一条边的水流都不能超过边权值（可以小于等于但不能大于），所以该图的最大流就是10+22+45=77。
- 加反向边

- 最小割
  - 最小割 = 最大流
- 最短路
  -

#6 矩阵连乘

- 给定N个矩阵$A_1, A_2, A_3 \ldots\ldots A_n$ 其中 A_i 和 A_{i+1} 可以相乘，其中A_i 的维度是$p[i-1] \times p[i]$ 如何确定矩阵连乘的计算次序，使得按照此次序计算该矩阵连乘所需要的数乘次数最少?
- $m[i][j]$ 表示从A_i 到 A_j 区间内最少乘的次数，枚举$k(i<k<j)$
- $m[i][j] = max(m[i][j], m[i][k] + m[k+1][j] + p[i-1] * p[k] * p[j])$

#7嵌套矩形

- 有n个矩形，每个矩形可以用a,b来描述，表示长和宽。矩形$X(a,b)$可以嵌套在矩形$Y(c,d)$中当且仅当$a<c, b<d$或者$b<c, a<d$（相当于旋转$X90$度）你的任务是选出尽可能多的矩形排成一行，使得除最后一个外，每一个矩形都可以嵌套在下一个矩形内。
- DAG上的dp
  - 先对矩形排序，长优先，宽次之，小到大
  - 然后跑dp

```
for(int i = 1;i < n; i++)
    for(int j = i + 1; j <= n; j++)
        if(a[i].x <a[j].x && a[i].y < a[j].y){
            if(dp[j] < 1 + dp[i]){
                dp[j] = 1 + dp[i];
                pre[j] = i; // 存储次序
            }
        }
```

#8任务安排

- $S=\{1,2,.....,n\}$ 个任务，$F=\{[S_i,F_i]\}$ 为任务的开始时间和结束时间
- 求最大相容集合，如何选择任务，才能执行最多的任务。
- 每次选F_i 最小的活动，也就是结束时间最小的活动，使能够余下更多的时间选择更多的活动。

#9区间调度问题

- 不带权
  - 跟任务安排一样
- 带权
  - 先按照结束时间非降序排序
  - 定义$p[i]$ 与任务i相容的最大下标
  - $opt[i]$ 前i个任务安排的最大权值
    - 如果 选择第i个，那么他一定是他前$p[i]$个任务的最优解+$w[i]$ 即 $opt[i]=opt[p[i]]+w[i]$
    - 如果不选，就是前 $i-1$ 个任务的最优解，$opt[i]=opt[i-1]$
  - $opt[i]=max(opt[i-1],opt[p[i]]+w[i])$

#10硬币问题

- 给你一个数组$coins[]$ 表示不同面额的硬币，以及一个$amount$ 表示总金额，计算凑成总金额的最少硬币个数，硬币数量无限
  - 本质是背包问题
  - $ans[j]=min(ans[j],ans[j-v[i]]+1)$