- 课程总结
- 习题总结
- 考试相关

# 近似 & 概率

# 课程总结

使用何种计算机，多少时间都不行！

问题

- 不可解问题（停机问题）
- 可解问题
  - 易解问题（多项式时间）
  - 难解问题（超多项式时间）

近似算法：难解→易解

常规计算机可以解决但耗费巨大！

# 课程总结


难解问题 的 **难度等价性**

**多一归约**

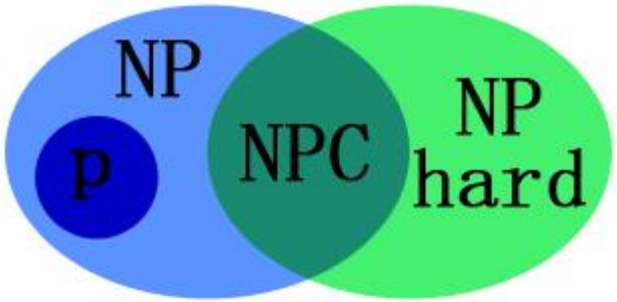
假设$L_1$和$L_2$是两个判定问题，f将$L_1$的每个实例I变换成$L_2$的实例f(I)。若对$L_1$的每个实例I，I的答案为“yes”当且仅当f(I)是$L_2$的答案为“yes”的实例，即：

$\forall I$，I是$L_1$输出“yes“的实例 $\Leftrightarrow$ f(I)是$L_2$输出”yes“的实例

则称f是从$L_1$到$L_2$的多一归约，记作：$L_1 \leqslant_m L_2$（传递关系）

**直观意义**：将求解$L_1$的问题转换为求解$L_2$的问题，而问题$L_1$不会难于$L_2$，$L_2$至少比$L_1$难。

# 课程总结




常见问题类型 的 **关系图**

**P问题：** 能在多项式时间内解决的问题

**NP问题：** 能在多项式时间验证答案正确与否的问题

**NPC问题**：对于一个(判定性)问题q，若

(1)q∈NP

(2)NP中任一问题均可多项式时间多一归约到q

则称问题q为NP-完全的(NP-complete，NPC)

**NP-hard问题**：若问题q仅满足条件(2)而不一定满足条件(1)，则问题q称为NP-难的(NP-hard，NPH)。显然：NPC⊆NP-hard

# 课程总结

问题  判断类型

- 证明q是**P问题**，找到多项式求解算法
- 证明q是**NP问题**，找到多项式验证算法
- 证明q是**NPH问题**，只需要将任一NPH问题多项式规约到q即可
- 证明q是**NPC问题**，在以上基础上再证明q是NP问题即可

## 课程总结

难解问题  需要**近似算法**

绝对性能度量 一个近似算法是优化问题Π的多项式时间近似算法A，使得对某一常数k>0满足：

$$\forall I \in D, \quad |A(I)-OPT(I)| \le k$$

相对性能度量 一个近似算法是优化问题Π的多项式时间近似算法A，使得对某一常数k>0满足：

$$\forall I \in D, \quad A(I)/OPT(I) \le k$$ ，最小化问题

$$OPT(I)/A(I) \le k$$ ，最大化问题

# 课程总结

## 四道例题

### 1.2 经典证明题

- Graph Coloring顶点着色问题 page50
- Knapsack背包问题 page54
- Scheduling 多机调度问题 page62
- Enclidean Travelling Salesman Problem 旅行商问题 page77

# 习题总结

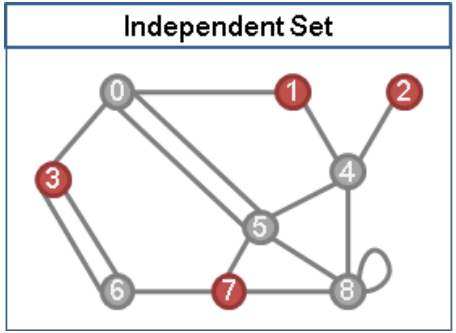
Ex. 1：给出与图的独立集问题相关的判定问题的形式化描述，并证明它是NP完全的（提示：团问题规约到独立集）

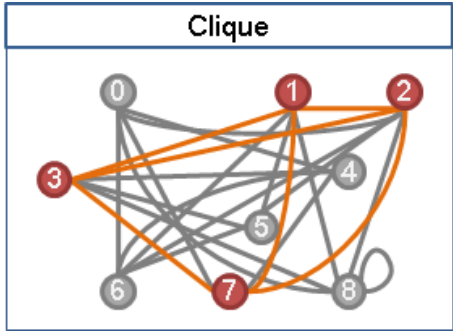
形式化：图G=(V,E)中存在子集V'∈V，任意u，v∈V'，(u, v)∉E

规约：给定$G=(V,E)$，团问题实例$<G,k>$

多项式时间计算得到$<\bar{G},k>$，$\bar{G}=(V,\bar{E})$

＝在$<\bar{G},k>$上的独立集问题

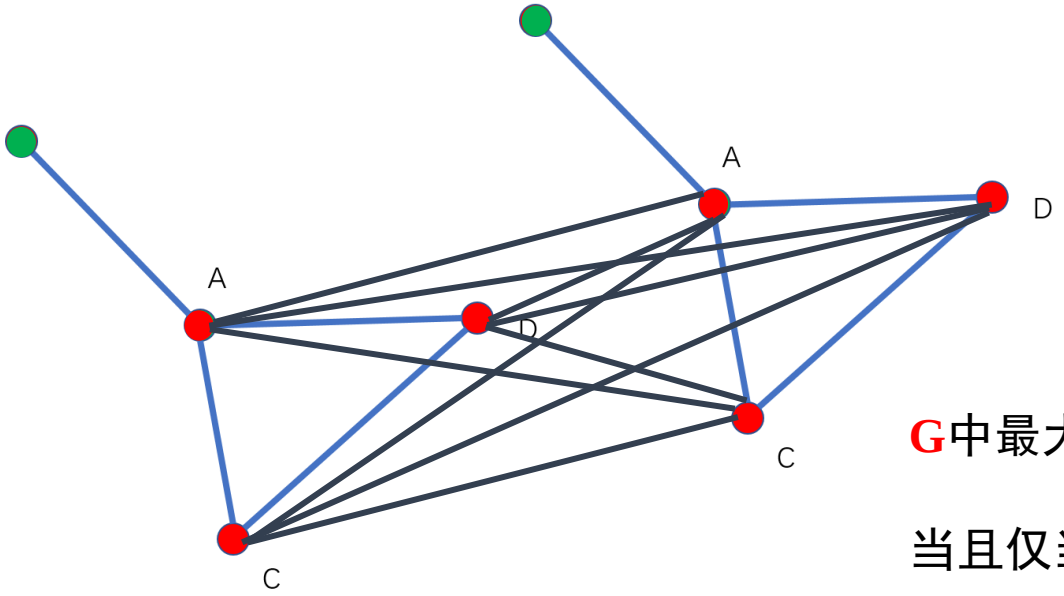


# 习题总结

图的m次幂$G^m$：取G的m个拷贝，连接位于不同副本里的任意两顶点

Ex. 2 ： G中最大团的size为α当且仅当$G^m$里最大团的size是mα




G中最大团的size为3

当且仅当$G^2$里最大团的size是6

# 习题总结

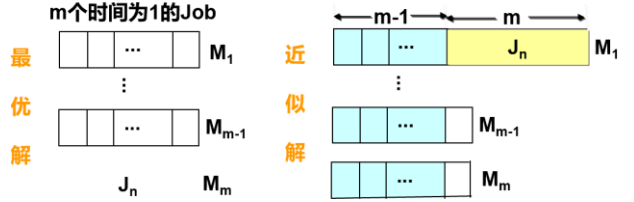
Ex. 3：完善证明 Th.1.9：**LPT算法的性能（近似）比：** $R_{LPT} = \frac{4}{3} - \frac{1}{3m}$

第一步：$\frac{A(I)}{OPT(I)} \le \frac{4}{3} - \frac{1}{3m}$ PPT p62-66 已证明

第二步：近似比的紧确界，给出实例 待证明

**§1.3.1 多机调度**

Pf(续)：现证近似比的紧确界。考虑输入实例$I^*$，设
$n = m(m-1)+1$
设前n-1个作业每个均有运行时间1，而最后1个作业$J_n$有$P_n$=m。易于证明：$OPT(I^*)=m$，而$A(I^*)=2m-1$
故有：$A(I^*)/OPT(I^*)=2-1/m$



# 考试相关

- 考试**形式**
  - **闭卷**

- **范围**
  - 概率算法（≈35）+近似算法（≈15） +分布式算法（≈50）

- **题型**
  - 选择题、问答题（论述、证明）+算法设计题（**参考题型**）

- **时间待定。。。**

# 课程总结

- 概率算法：算法加入随机性，可能要比确定的算法效果要好。
- 对概率算法的评价：使用期望时间评估概率算法的时间。

- 概率算法分类：
- Numerical
- Monte Carlo
- Las Vegas
- Sherwood

# 课程总结

- Numerical-数字概率算法

- **特点：执行时间越长，精确度越高**
- 求解数学问题的近似解。
- 投镖法 或者 积分区间均匀取点近似。
- 例题：积分求解，求$\pi$值；概率计数 – 计算set 中的势；集合中不同对象的计数

# 课程总结

- Sherwood 算法

- **特点：用于减少甚至消除好的样例和坏的样例之间的差别。**
- 加入随机性，使得运行时间不再和实例相关。（不再有最坏的实例，但是仍有最坏的运行时间。）
- 流程:
- ①将被解的实例变换到一个随机实例。// 预处理
- ②用确定算法解此随机实例，得到一个解。
- ③将此解变换为对原实例的解。 // 后处理

# 课程总结

- Sherwood 算法

① $(\forall n \in N)(\forall x, y \in X_n)(\exists! r \in A_n)[u(x,r) = y]$

此性质说明原实例x可通过随机抽样变换成另一个实例 y

∃！表示存在且只存在一个

② $(\forall n \in N)(\forall x \in X_n)(\forall r \in A_n)[f(x) = v(r, f(u(x,r)))]$

此性质表示对$y$的解可变换为对原实例$x$的解

③ 函数**u**和**v**在最坏情况下能够有效计算

- 作业题：能清楚知道u和v分别是什么。或者给你一个问题能不能给出u和v，来随机化样例，使得时间复杂度和输入无关。
- 例题：搜索有序表；离散对数计算；排序

# 课程总结

- Las Vegas 算法

- **特点：获得的答案必定正确，但不确保能获得答案**
- 使用同一实例多次运行，有独立机会获得解。成功概率随时间增加而增加。
- 例题：8皇后问题，**模p 平方根，整数的因式分解**

**设t(x)是算法obstinate找到一个正确解的期望时间，则**

$$t(x) = p(x)s(x) + (1 - p(x))(e(x) + t(x))$$

$LV$成功的概率　LV失败的概率

$$t(x) = s(x) + \frac{1 - p(x)}{p(x)} e(x)$$

**若要最小化t(x)，则需在p(x)，s(x)和e(x)之间进行某种折衷，例如，若要减少失败的时间，则可降低成功的概率p(x)。**

# 课程总结

- Monte Carlo 算法

- **特点：适用于问题只有一个正确解，没有近似解。（判定问题）**
- **MonteCarlo 算法偶尔会犯错，但是对所有实例都是以高概率返回正确解。**
- 概念：MC 算法 **P 正确的**；MC 算法 **相容**。
- 例题：**有偏算法**– 主元素问题；素数判定问题；矩阵乘法验证

# 习题

Ex1.若将y ← uniform(0, 1) 改为 y ← x, 则上述的算法估计的值是什么?

- 即函数在求$2x^2 \leq 1$ 时 k++，即$x \leq \frac{\sqrt{2}}{2}$; 而x ← uniform(0, 1) 因此 $\frac{\mathrm{k}}{\mathrm{n}} = \frac{\sqrt{2}}{2}$ 最终返回的结果即为 $2\sqrt{2}$

# 习题

Ex2.在机器上用$4\int_0^1 \sqrt{1-x^2}dx$估计Π值，给出不同的n值和相应的精度

```
for(int j=0; j<N; j++){
        double x = rand() / double(RAND_MAX);
        double y = rand() / double(RAND_MAX);
        if(pow(x, 2) + pow(y, 2)<=1)  y<=f(x) 变换一下
              k++;
        }
        cout << “N = ” <<N <<”, pi= ” << (k*4.0)/N << endl;
}
```

执行结果：

N = 10000, pi=3.142

N = 100000, pi = 3.14104

N = 1000000, pi = 3.14189

N = 10000000, pi = 3.14154

N = 100000000, pi = 3.14132

# 习题

Ex3.设a, b, c和d是实数，且a ⩽ b, c ⩽ d, f:[a, b] → [c, d]是一个连续函数，写一概率算法计算积分:$\int_a^b f(x)dx$ 。其中函数参数为a,b,c,d,n,f。其中f用函数指针实现。

```
double cacl(double a, double b, double c, double d, unsigned n, func f){
    unsigned int k = 0;
    for(int i=0; i < n; i++){
        double x = a + rand() / double(RAND_MAX) * (b-a);
        double y = c + rand() / double(RAND_MAX) * (d-c);
        if(y <= f(x))
            k++;
    }
    Return k*(b-a)*(d-c)/n;
}
```

# 习题

Ex4.用上述算法，估计整数子集1~n的大小，并分析n对估计值的影响。

```
import random
import math


def SetCount(X):
    k = 0
    times = 1000
    for i in range(times):
        S = []
        a = random.randint(1,X);
        while a not in S:
            k+=1
            S.append(a)
            a = random.randint(1,X)
    k /= times
    return 2*k*k/math.pi
```

这个方法整体下来误差还是挺大的
随着n的变化误差都差的不多

对k值取平均再运算会比每次都计算$2\frac{k^2}{\pi}$再取平均得到的误差更小

# 习题

Ex5. 分析dlogRH的工作原理，指出该算法相应的u和v

- dlogRH 的伪代码：
- dlogRH(g, a, p) { // 求$log_{g,p}a$, a = $g^x$ mod p， 求x
  // Sherwood算法
  r ← uniform(0..p-2);
  b ← ModularExponent(g, r, p); //求幂模b=gr mod p
  c ← ba mod p; //(($g^r$ mod p)($g^x$mod p))mod p=$g^{r+x}$mod p=c
  y ← $log_{g,p}c$ ; // 使用确定性算法求$log_{g,p}a$, y=r+x
  return (y-r) mod (p-1); // 求x
  }

在求$log_{g,p}a$上多增加了一个Sherwood 算法的外壳，减少最坏例子的影响。
1.第一步随机选取r，实现Sherwood算法的随机取元素操作。
2.实例随机化，将原实例a转化为随机实例c
3.用确定算法求y的解s 。并转回原所求a

u是 `c = ba mod p` 这一步（将a变化为c，即 `x` 变换 `x+r`)
v是 `(y-r) mod (p-1)` 这一步（对解的逆变换)

# 习题

Ex6. 写一Sherwood算法C，与算法A, B, D比较，给出实验结果。

- 在算法 B 的基础上修改，不是取前 sqrt(n) 个元素作为基准，而是随机取 sqrt(n) 个元素作为基准。

- 多次执行的结果和算法B 差别不大 。

# 习题

Ex7.证明：当放置（k+1)th皇后时，若有多个位置是开放的,则算法QueensLV选中其中任一位置的概率相等

- 对于第$n_b$个可放置皇后来说，其被选中的概率为 $\frac{1}{n_b}$;
- 前一个被选中的概率为$\frac{1}{n_b-1}$，其后续不被替换掉的可能性为 $\frac{n_b-1}{n_b}$。
- 因此倒数第二个成立元素最终得以保留的概率为 $\frac{1}{n_b-1}\times\frac{n_b-1}{n_b}=\frac{1}{n_b}$;
- 依次类推，第一个皇后选中概率为$1\times\frac{1}{2}\times\frac{2}{3}\times\cdots\times\frac{n_b-1}{n_b}=\frac{1}{n_b}$。因此所有的开放位置的概率相等。

# 习题

Ex8. 写一算法，求n=12~20时最优的StepVegas值。

```
def BackTrace(res, k, step_vegas, col, diag45, diag135):
    n,j,k0 = len(res),0,k
    nodes = 0
    while k <= n-1:
        flag = False
        for i in range(j,n):
            if i not in col and i-k-1 not in diag45 and i+k+1 not in diag135:
                nodes += 1
                k+=1
                res[k-1] = i
                col.append(i)
                diag45.append(i-k)
                diag135.append(i+k)
                flag = True
                j = 1
                break
        if not flag:
            k -= 1
            if k < k0:
                return False,nodes
            col.pop()
            diag45.pop()
            diag135.pop()
            j = res[k]+1
    return True,nodes
```

```
def StepVegas(n, step_vegas):
    col,diag45,diag135 = [],[],[]
    res = [0] * n
    k,nb = 0,1
    nodes = 0
    while not(nb == 0 or k == step_vegas):
        nb = 0
        for i in range(1,n+1):
            if i not in col and i-k-1 not in diag45 and i+k+1 not in diag135:
                nb += 1
                if random.randint(1,nb) == 1:

                    j = i
        if nb > 0:
            k += 1
            res[k-1] = j
            nodes += 1
            col.append(j)
            diag45.append(j-k)
            diag135.append(j+k)
    if nb > 0:
        success,back_nodes = BackTrace(res, k+1, step_vegas, col, diag45, diag135)
        return success,back_nodes + nodes
    else:
        return False,nodes
```