As this is a maximization problem, we need an upper bound of c^*, and there is an easy one:

$$c^* \leq m$$

where $m = |E|$.

The algorithm is: coloring every node independently with one of the three colors, each with probability $\frac{1}{3}$.

Let random variable

$$X_e = \begin{cases} 1 & \text{edge } e \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

Then for any given edge e, there are 9 ways to color its two ends, each of which appears with the same probability, and 3 of them are not satisfying.

$$Exp[X_e] = Pr[e \text{ is satisfied}] = \frac{6}{9} = \frac{2}{3}$$

Let Y be the random variable denoting the number of satisfied edges, then by linearity of expectations,

$$Exp[Y] = Exp\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} Exp[X_e] = \frac{2}{3}m \geq \frac{2}{3}c^*$$

1. 三着色是一个 yes-no 问题，但我们可以如下把它作为一个优化问题.

任给一个图 $G=(V,E)$，要给每一个结点着 3 种颜色中的一种颜色，但不要求每一对相邻的结点必须着不同的颜色. 如果一条边 (u,v) 的两个端点着不同的颜色，则称这条边是满足的.

考虑使满足的边数最大的三着色，并用 c^* 表示这个最大的边数. 试给一个多项式时间算法，要求它生成至少满足 $\frac{2}{3}c^*$ 条边的三着色. 如果需要的话，算法可以是随机的，这时它满足的期望边数必须不小于 $\frac{2}{3}c^*$.

[1] ex568.721.313

Number the voters $1, 2, \ldots, 100,000$, where voters 1 through 20000 are the Republican voters. Let X_i be the random variable equal to 0 if i votes for R, and 1 if i votes for D. So $X = \sum_{i=1}^{100000} X_i$.

Now, for $i \leq 20000$, $EX_i = .99 \cdot 0 + .01 \cdot 1 = .01$. For $i > 20000$, $EX_i = .01 \cdot 0 + .99 \cdot 1 = .99$. By linearity of expectation,

$$EX - \sum_{i=1}^{100000} EX_i - 20000 \cdot .01 + 80000 \cdot .99 - 79400.$$

2. 考虑一个有 10 万选民的县. 选票上只有两名候选人：一名民主党候选人(用 D 表示)和一名共和党候选人(用 R 表示). 在选举的时候这个县严重地倾向于民主党，有 8 万人打算投 D 的票，2 万人打算投 R 的票.

但是，选票的设计出了一点问题，使得每一个选举人独立地以 1/100 的概率投错候选人——也就是说，他或她投给了不打算投的人. (记住在这次选举中选票上只有两名候选人.)

设随机变量 X 等于当选举如此进行时民主党候选人 D 的得票数. 试确定 X 的期望值，并解释这个值的来源.

[1] ex734.264.279

(a) Assume that using the described protocol, we get a set S that is not conflict free. Then there must be 2 processes P_i and P_j in the set S that both picked the value 1 and are going to want to share the same resource. But this contradicts the way our protocol was implemented, since we selected processes that picked the value 1 and whose set of conflicting processes all picked the value 0. Thus if P_i and P_j both picked the value 1, neither of them would be selected and so the resulting set S is conflict free. For each process P_i, the probability that it is selected depends on the fact that P_i picks the value 1 and all its d conflicting processes pick the value 0. Thus $P[P_i\text{selected}] = \frac{1}{2} * (\frac{1}{2})^d$. And since there are n processes that pick values independently, the expected size of the set S is $n * (\frac{1}{2})^{d+1}$

(b) Now a process P_i picks the value 1 with probability p and 0 with probability $1-p$. So the probability that P_i is selected (i.e. P_i picks the value 1 and its d conflicting processes pick the value 0) is $p * (1-p)^d$. Now we want to maximize the probability that a process is selected. Using calculus, we take the derivative of $p(1-p)^d$ and set it equal to 0 to solve for the value of p that gives the objective it's maximum value. The derivative of $p(1-p)^d$ is $(1-p)^d - dp(1-p)^{d-1}$. Solving for p, we get $p - \frac{1}{d+1}$. Thus the probability that a process is selected is $\frac{d^d}{(d+1)^{d+1}}$ and the expected size of the set S is $n * \frac{d^d}{(d+1)^{d+1}}$. Note that this is $\frac{n}{d}$ times $(1 - \frac{1}{d+1})^{d+1}$ and this later term is $\frac{1}{e}$ in the limit and so by changing the probability, we got a fraction of $\frac{n}{d}$ nodes. Note that with $p = 0.5$, we got an exponentially small subset in terms of d.

3. 在 13.1 节，我们看到一个解决特定消除争用问题的简单分布式协议. 这里有另一种场合，在这个场合中随机化借助一个独立集的分布式构造能够有助于消除争用.

假设一个有 n 个进程的系统. 某些进程之间有冲突：它们都要求访问一个共享的资源. 在给定的时间段，目标是安排一个大的进程子集 S 运行——其余的进程不运行——使得在这个被安排的子集 S 中任何两个进程都不冲突. 我们称这样的子集 S 是无冲突的.

可以用图 $G=(V,E)$ 描绘这个系统，一个结点表示一个进程，一条边连接一对冲突的进程. 容易验证进程集合 S 是无冲突的当且仅当它构成 G 中的一个独立集. 这意味着对任意的冲突 G，求最大的无冲突集合 S 将是困难的(因为一般的独立集问题可归约到这个问题). 虽然如此，我们仍可以发现找到合理大的无冲突集合的启发式算法. 而且我们可能喜欢没有集中控制就能到达这个目的的简单方法：每一个进程只需要与少数其他进程通信并决定它是否应该属于集合 S.

对这个问题，我们假设在图 G 中每一个结点恰好有 d 个邻点(即，每一个进程恰好与 d 个其他的进程冲突).

(a) 考虑下述简单的协议.

每一个进程 P_i 独立地取一个随机值 x_i，x_i 以概率 1/2 等于 1，以概率 1/2 等于 0. 然后，它决定进入集合 S 当且仅当它取 1 并且与它冲突的所有进程都取 0.

试证明执行这个协议所得到的集合 S 是无冲突的. 并且给出用 n(进程数)和 d(每一个进程的冲突数)表示的 S 的期望大小的公式.

(b) 在上面的协议中选择概率 1/2 是相当任意的，并且不清楚它是否一定给出最好的系统性能. 协议更一般的规范用 0 与 1 之间的参数 p 代替概率 1/2，具体做法如下.

每一个进程 P_i 独立地取一个随机值 x_i，x_i 以概率 p 等于 1，以概率 $1-p$ 等于 0. 然后，它决定进入集合 S 当且仅当它取 1 并且与它冲突的所有进程都取 0.

给出 p 的值使所得到的集合 S 的期望大小尽可能的大，用图 G 的参数表示. 给出当 p 取这个最优值时 S 的期望大小的表达式.

[1] ex131.386.529

4. 因特网上的若干对等系统以层叠网为基础. 这些系统不使用物理的因特网拓扑作为完成计算的网络，而是运行结点选择虚拟“邻点”集合的协议，这样定义一个高层次的图，它的构造可能与基础的物理网络没有关系. 这样的层叠网用于共享数据和服务，与物理网络相比它极其灵活，物理网络很难实时地修改以适应变化的条件.

对等网络随着新的参加者的到来通常会增长，他们通过链入现存的构造加入这个系统. 这个增长过程对整个网络的特征有着固有的影响. 最近人们已经研究了一种简单抽象的网络增长模型，这种模型使我们能够定性地深入观察在现实的网络中这个过程的行为方式.

下面是这种模型的一个简单例子. 系统从一个结点 v_1 开始. 然后，每次加入一个结点. 当一个结点加入时，它执行协议，因而形成它到另一个结点的链接，这个结点是从系统的已有结点中等可能随机选择的. 更具体地，设系统已经包含结点 $v_1, v_2, \cdots, v_{k-1}$，结点 v_k 要加入，那么它随机地选择 $v_1, v_2, \cdots, v_{k-1}$ 中的一个并链接到这个结点.

假设我们运行这个过程直至得到一个包含结点 $v_1, v_2, \cdots, v_n$ 的系统为止，上面描述的随机过程将产生一个有向网络，在这个网络中除 v_1 之外的每一个结点恰好有一条输出边. 另一方面，一个结点可能有多条输入链接或者一条也没有. 到结点 v_j 的输入链接反映所有通过 v_j 访问这个系统的其他结点，因此如果 v_j 有许多输入链接，那么它可能被安置了很大的负载. 为了保持系统的负载均衡，将希望所有的结点有大致相同数目的输入链接. 但是，这不大可能发生，因为在这个过程中较早加入的结点比较晚加入的结点可能有更多的输入链接. 我们试图如下量化这种不均衡.

(a) 给定上面描述的随机过程，在所得到的网络中到结点 v_j 的输入链接的期望数是多少？ 给出用 n 和 j 表示的精确公式，并用 $\Theta(\cdot)$ 记号渐近地(用不含大的求和式的表达式)表示这个量.

(b) 部分(a)精确地表达了在网络中早到的结点担负“不公平的”链接分配的意思. 另一种量化这种不均衡的方法是，观察到在这个随机过程的运行中，我们预料许多结点最终没有输入链接.

给出在按照这个模型随机增长的网络中没有输入链接的结点的期望数的公式.

(a) For every node v_k that comes later than v_j, i.e. $k > j$, it has probability $\frac{1}{k-1}$ to link to v_j, since v_k chooses from the $k-1$ existing nodes with equal probabilities. For all the nodes coming before v_j, such probability is obviously zero.

So the expected number of incoming links to node v_j is

$$\begin{aligned}\sum_{k=j+1}^{n} \frac{1}{k-1} &= \sum_{k=1}^{n-1} \frac{1}{k} - \sum_{k=1}^{j-1} \frac{1}{k} \\ &= H(n-1) - H(k-1) \\ &= \Theta(\ln n) - \Theta(\ln k) \\ &= \Theta(\ln \frac{n}{k})\end{aligned}$$

(b) Consider a node v_j, every node v_k with $k > j$ has probability $1 \quad \frac{1}{k-1}$ not to link to v_j. So if we have random variable X_j s.t.

$$X_j = \begin{cases} 1 & \text{node } v_j \text{ has no in-coming links} \\ 0 & \text{otherwise} \end{cases}$$

then

$$\begin{aligned}Exp[X_j] &= Pr[\text{no nodes links to } v_j] \\ &= \prod_{k=j+1}^{n} \left(1 - \frac{1}{k-1}\right) \\ &= \frac{j-1}{j} \cdot \frac{j}{j+1} \cdot \frac{j+1}{j+2} \cdots \frac{n-2}{n-1} \\ &= \frac{j-1}{n-1}\end{aligned}$$

Therefore, by linearity of expectations, we get the expected number of nodes without in-coming links

$$\sum_{j=1}^{n} Exp[X_j] = \sum_{j=1}^{n} \frac{j-1}{n-1} = \frac{1}{n-1} \sum_{j=1}^{n} (j-1) = \frac{1}{n-1} \cdot \frac{n(n-1)}{2} = \frac{n}{2}$$

[1] ex976.627.720

5. 在一个县的某处偏远乡村有 n 个小镇，它们决定用大容量光缆连接到一个因特网交换中心。把这些小镇分别记作 $T_1, T_2, \cdots, T_n$，它们沿着一条高速公路依次排列，小镇 T_i 离交换中心 i 哩（见图 13.6）。

图 13.6　小镇 $T_1, T_2, \cdots, T_n$ 必须决定如何分摊光缆的费用

这种光缆相当贵，每哩 k 美元，结果是整条光缆的总费用为 kn 美元。这 n 个小镇集合在一起讨论如何分摊光缆的费用。

首先，位于高速公路最远端的小镇提出下述建议。

建议 A：所有的小镇平均分摊费用，每一个镇支付 k 美元。

在某种意义下建议 A 是公平的，因为这好像每一个小镇支付直接接入它的那一哩光缆的费用。

但是，一个离交换中心很近的小镇反对，指出离得远的小镇实际上使用一大段光缆，而离得近的小镇只使用一小段光缆。因此他们提出下述反建议。

建议 B：小镇 T_i 分摊的费用正比于 i，即它到交换中心的距离。

另一个离交换中心很近的小镇指出另一种非正比的分摊办法也是自然的。这个办法的基础是把整条光缆分成 n 条长度相等的“边” $e_1, e_2, \cdots, e_n$，其中第一条边 e_1 从交换站到 T_1，第 i 条边 $e_i (i>1)$ 从 T_{i-1} 到 T_i。现在我们发现所有的小镇都使用 e_1，而只有最远的小镇使用 e_n。因此，他们提出

建议 C：把这个费用分到每一条边 e_i 上，小镇 $T_i, T_{i+1}, \cdots, T_n$ 应同样地负担边 e_i 的费用，因为它们是 e_i “下游”的小镇。

于是，现在有许多不同的意见，哪一个是公平的？为了解决这个问题，他们求助于 Lloyd Shapley 的成果，这是一位 20 世纪最著名的数学经济学家。他提出现在称做 Shapley 值的概念作为在几个部分之间分摊成本或利润的一般性机制。可以把它看做在假设各部分按随机顺序到达的条件下，确定每一个部分的“边际贡献”。

在这个问题中这样具体应用这个概念。考虑小镇的排列顺序 $\mathcal{O}$，并假设小镇按这个顺序“到达”。如下确定在顺序 $\mathcal{O}$ 下小镇 T_i 的边际费用。如果在顺序 $\mathcal{O}$ 中 T_i 是第一个，那么 T_i 支付 ki，即从交换中心到 T_i 的全部光缆的费用。否则，看在顺序 $\mathcal{O}$ 中 T_i 前面到达的所有小镇，设 T_j 是这些小镇中离交换中心最远的。我们假设当 T_i 到达时，光缆已经铺到 T_j、但没有再往前铺。于是，如果 $j>i$（T_j 比 T_i 更远），那么 T_i 的边际费用是 0，因为光缆在到 T_j 的路上已经经过 T_i。如果 $j<i$，那么 T_i 的边际费用是 $k(i-j)$：光缆从 T_j 延伸到 T_i 的费用。

（例如，设 $n=3$ 且小镇到达的顺序为 T_1, T_3, T_2。首先 T_1 当它到达时支付 k。接着，当 T_3 到达时，它只需要支付 $2k$ 把光缆从 T_1 延伸过来。最后，当 T_2 到达时，它不需支付任何费用，因为光缆已经经过它到达 T_3。）

现在设随机变量 X_i 等于当从小镇的所有排列中等可能地随机选择顺序 $\mathcal{O}$ 时小镇 T_i 的边际费用。按照 Shapley 值的规则，T_i 对光缆总费用应贡献的量是 X_i 的期望值。

问题是：上面提出的 3 个建议中哪一个（如果有的话）给出与 Shapley 值费用分摊机制相同的费用分摊？并给出证明。

Solution.

The Shapley value mechanism is equivalent to Proposal C. Under Proposal C, town T_i pays

$$\sum_{j=1}^{i} \frac{k}{n-j+1},$$

since there are $n-j+1$ towns downstream of edge e_j, and so the contribution of T_i to edge e_j (for $1 \le j \le i$) is $k/(n-j+1)$.

Now, let X_i be the random variable in the question. We decompose X_i into a random variable corresponding to each of the edges: let Y_j be the random variable equal to the amount town T_i pays for edge e_j ($1 \le j \le i$). Note that $X_i = \sum_{j=1}^{i} Y_j$, and so by linearity of expectation, $EX_i = \sum_{j=1}^{i} EY_j$.

What is EY_j? With probability $1/(n-j+1)$, town T_i is the first among the towns downstream of e_j to arrive, and it incurs a cost of k. Otherwise, when town T_i arrives, some other town downstream of e_j has already arrived, and so it incurs a cost of 0. Thus,

$$EY_j = 0\left(1 - \frac{1}{n-j+1}\right) + k\left(\frac{1}{n-j+1}\right) = \frac{k}{n-j+1}.$$

Hence

$$EX_i = \sum_{j=1}^{i} \frac{k}{n-j+1},$$

which is the same as the contribution of T_i under Proposal C.

We interpret the constraint (μ_i, μ_j, μ_k) to mean that we require one of the subsequences $\ldots, \mu_i, \ldots, \mu_j, \ldots, \mu_k, \ldots$ or $\ldots, \mu_k, \ldots, \mu_j, \ldots, \mu_i, \ldots$ to occur in the ordering of the markers. (One could also interpret it to mean that just the first of these subsequences occurs; this will affect the analysis below by a factor of 2.)

Suppose that we choose an order for the n markers uniformly at random. Let X_t denote the random variable whose value is 1 if the t^{th} constraint (μ_i, μ_j, μ_k) is satisfied, and 0 otherwise. The six possible subsequences of $\{\mu_i, \mu_j, \mu_k\}$ occur with equal probability, and two of them satisfy the constraint; thus $EX_t = \frac{1}{3}$. Hence if $X = \sum_t X_t$ gives the total number of constraints satisfied, we have $EX = \frac{1}{3}k$.

So if our random ordering satisfies a number of constraints that is at least the expectation, we have satisfied at least $\frac{1}{3}$ of all constraints, and hence at least $\frac{1}{3}$ of the maximum number of constraints that can be simultaneously satisfied.

We can extend this to construct an algorithm that *only* produces solutions within a factor of $\frac{1}{3}$ of optimal: We simply repeatedly generate random orderings until $\frac{1}{3}k$ of the constraints are satisfied. To bound the expected running time of this algorithm, we must give a lower bound on the probability p^+ that a single random ordering will satisfy at least the expected number of constraints; the expected running time will then be at most $1/p^+$ times the cost of a single iteration.

First note that k is at most n^3, and define $k' = \frac{1}{3}k$. Let k'' denote the greatest integer strictly less than k'. Let p_j denote the probability that we satisfy j of the constraints. Thus $p^+ = \sum_{j \geq k'} p_j$; we define $p^- = \sum_{j < k'} p_j = 1 - p^+$. Then we have

$$\begin{aligned} k' &= \sum_j j p_j \\ &= \sum_{j<k'} j p_j + \sum_{j \geq k'} j p_j \\ &\leq \sum_{j<k'} k'' p_j + \sum_{j \geq k'} n^3 p_j \\ &= k''(1 - p^+) + n^3 p^+ \end{aligned}$$

from which it follows that

$$(k'' + n^3)p^+ \geq k' - k'' \geq \frac{1}{3}.$$

Since $k'' \leq n^3$, we have $p^+ \geq \frac{1}{6n^3}$, and so we are done.

6. 有许多来自基因组图谱的难问题，其中的一个可以用下述抽象的方式描述. 给定 n 个标记的集合 $\{\mu_1, \cdots, \mu_n\}$，它们是要绘制图谱的染色体上的位置，我们的目标是输出这些标记的线性顺序. 输出的顺序必须与 k 个限制一致，每一个限制由一个 3 元组 (μ_i, μ_j, μ_k) 给出，要求在生成的整个顺序中 μ_j 位于 μ_i 与 μ_k 之间（注意这个限制没有指定 μ_i 与 μ_k 中哪一个按这个顺序必须在前面，而只限制 μ_j 必须在它们之间）.

有时不可能同时满足所有的限制，因而我们希望产生一个顺序满足尽可能多的限制. 不幸的是，确定是否有一个顺序至少满足 k 个限制中的 k' 个是一个 NP 完全问题（你不需要证明）.

试给出一个常数 $\alpha > 0$（与 n 无关）和一个具有下述性质的算法. 如果能够满足 k^* 个限制，那么算法生成的标记顺序至少满足 αk^* 个限制. 你的算法可以是随机的，这时它必须生成一个满足的期望限制数至少为 αk^* 的顺序.

(a) Consider a clause C_i with n variables. The probability that the clause is not satisfied is $\frac{1}{2^m}$ and so the probability that it is satisfied is 1 less this quantity. The worst case is when C_i has just one variable, i.e. $n = 1$, in which case the probability of the clause being satisfied is $\frac{1}{2}$. Since there are k clauses, the expected number of clauses being satisfied is at least $\frac{k}{2}$. Consider the two clauses x_1 and $\overline{x_1}$. Clearly only one of these can be satisfied.

(b) For variables that occur in single variable clauses, let the probability of setting the variable so as to satisfy the clause be $p \geq \frac{1}{2}$. For all other variables, let the probabilities be $\frac{1}{2}$ as before. Now for a clause C_i with n variables, $n \geq 2$, the probability of satisfying it is at worst $(1 - \frac{1}{2^n}) \geq (1 - p^2)$ since $p \geq \frac{1}{2}$. Now to solve for p, we want to satisfy all clauses, so solve $p = 1 - p^2$ to get $p \approx 0.62$. And hence the expected number of satisfied clauses is $0.62n$.

(c) Let the total number of clauses be k. For each pair of single variable conflicting clauses, i.e. x_i and $\overline{x_i}$, remove one of them from the set of clauses. Assume we have removed m clauses. Then the maximum number of clauses we could satisfy is $k - m$. Now apply the algorithm described in the previous part of the problem to the $k - 2m$ clauses that had no conflict to begin with. The expected number of clauses we satisfy this way is $0.62 * (k - 2m)$. In addition to this we can also satisfy m of the $2m$ conflicting clauses and so we satisfy $0.62 * (k - 2m) + m \geq 0.62 * (k - m)$ clauses which is our desired target. Note that this algorithm is polynomial in the number of variables and clauses since we look at each clause once.

7. 在13.4节我们设计了 MAX 3-SAT 问题的一个近似到 7/8 因子的近似算法，在那里我们假设每一个子句有与 3 个不同变量相关联的项. 现在要考虑一个类似的 MAX SAT 问题：给定变量集 $X=\{x_1,\cdots,x_n\}$ 上的一组子集 $C_1,\cdots,C_k$，求一个真值赋值满足尽可能多的子句. 每一个子句至少有一个项，一个子句中的所有变量是不同的，除此之外对子句的长度没有做任何假设：可能有一些子句有很多变量，而另一些可能只有一个变量.

(a) 首先考虑我们用于 MAX 3-SAT 的随机近似算法，以概率 1/2 独立地令每一个变量为真或假. 证明这个随机赋值满足的期望子句数不小于 $k/2$，即所有的子句中至少期望地有一半被满足. 给出一个例子说明存在 MAX SAT 实例使得任何赋值满足的子句都不超过所有子句的一半.

(b) 如果有一个只由一项构成的子句(例如，仅由 x_1 或仅由 $\overline{x}_2$ 构成的子句)，那么只有唯一的方法满足它：必须以适当的方式给对应的变量赋值. 如果有两个子句，其中一个仅由 x_i 构成，而另一个仅由它的否定项 $\overline{x}_i$ 构成，那么这是一个相当直接的矛盾.

假设实例不含这种"冲突的子句"对，即没有变量 x_i 使得既有一个子句 $C=\{x_i\}$、又有一个子句 $C'=\{\overline{x}_i\}$. 修改上面的随机算法把近似因子从 1/2 改进到 0.6. 即，修改算法使得被满足的期望子句数至少为 $0.6k$.

(c) 试给一个一般 MAX SAT 问题的多项式时间随机算法，使得算法满足的期望子句数不少于最大可能的 0.6.

(注意，根据部分(a)中的例子，存在不可能满足多于 $k/2$ 个子句的实例. 这里的要点是仍可指望得到一个有效的算法，它能够期望地满足最优赋值能够满足的最大子句数的 0.6.)

First we give an algorithm that produces a subgraph whose expected number of edges has the desired value. For this, we simply choose k nodes uniformly at random from G. Now, for $i < j$, let X_{ij} be a random variable equal to 1 if there is an edge between our i^{th} and j^{th} node choices, and equal to 0 otherwise.

Of the $n(n-1)$ choices for i and j, there are $2m$ that yield an edge (since an edge (u, v) can be chosen either by picking u in position i and v in position j, or by picking v in position i and u in position j). Thus $E[X_{ij}] = \frac{2m}{n(n-1)}$.

The expected number of edges we get in total is

$$\sum_{i<j} E[X_{ij}] = \binom{k}{2} \cdot \frac{2m}{n(n-1)} = \frac{mk(k-1)}{n(n-1)}.$$

We now want to turn this into an algorithm with expected polynomial running time, which always produces a subgraph with at least this many edges. The analogous issue came up with MAX 3-SAT, and we use the same idea here: For this we use the same idea as in the analogous MAX 3-SAT: we run the above randomized algorithm repeatedly until it produces a subgraph with at least the desired number of edges.

Let p^+ be the probability that one iteration of this succeeds; our overall running time will be the (polynomial) time for one iteration, times $1/p^+$. First note that the maximum number of edges we can find is $e = \frac{k(k-1)}{2}$, and we're seeking $e' = e \cdot \frac{2m}{n(n-1)}$. Let e'' denote the greatest integer strictly less than e'. Let p_j denote the probability that we find a subgraph with exactly j edges. Thus $p^+ = \sum_{j>e'} p_j$; we define $p^- = \sum_{j<e'} p_j = 1 - p^+$. Then we have

$$\begin{aligned} e' &= \sum_j j p_j \\ &= \sum_{j<e'} j p_j + \sum_{j \geq e'} j p_j \\ &\leq \sum_{j<e'} e'' p_j + \sum_{j \geq e'} e p_j \\ &= e''(1-p^+) + \binom{k}{2} p^+ \end{aligned}$$

from which it follows that

$$(e'' + \binom{k}{2}) p^+ \geq e' - e'' \geq \frac{1}{n(n-1)}.$$

Since $e'' \leq \binom{k}{2}$, we have $p^+ \geq \frac{1}{k(k-1)n(n-1)}$, and so we are done.

8. 设 $G=(V,E)$ 是 n 个结点和 m 条边的无向图. 对子集 $X \subseteq V$, 用 $G[X]$ 表示在 X 上诱导出的子图, 即结点集合为 X、边集合由 G 中所有两个端点都在 X 中的边组成的图.

给定一个自然数 $k \leqslant n$, 要找一个诱导出 G 的"稠密"子图的 k 个结点的集合, 具体的意思如下. 给出一个多项式时间算法, 它对给定的自然数 $k \leqslant n$, 生成 k 个结点的集合 X 使得诱导子图 $G[X]$ 至少有 $\frac{mk(k-1)}{n(n-1)}$ 条边.

你可以或者(a) 给出一个确定的算法, 或者(b) 给出一个随机算法, 它的期望运行时间是多项式的, 且只输出正确的答案.

The strategy is as follows. The seller watches the first $n/2$ bids without accepting any of them. Let b^* be the highest bid among these. Then, in the final $n/2$ bids, the seller accepts any bid that is larger than b^*. (If there is no such bid, the seller simply accepts the final bid.)

Let b_i denote the highest bid, and b_j denote the second highest bid. Let S denote the underlying sample space, consisting of all permutations of the bids (since they can arrive in any order.) So $|S| = n!$. Let E denote the event that b_j occurs among the first $n/2$ bids, and b_i occurs among the final $n/2$ bids.

What is $|E|$? We can place b_j anywhere among the first $n/2$ bids ($n/2$ choices); then we can place b_i anywhere among the final $n/2$ bids ($n/2$ choices); and then we can order the remaining bids arbitrarily ($(n-2)!$ choices). Thus $|E| = \frac{1}{4}n^2(n-2)!$, and so

$$P[E] = \frac{n^2(n-2)!}{4n!} = \frac{n}{4(n-1)} \geq \frac{1}{4}.$$

Finally, if event E happens, then the strategy will accept the highest bid; so the highest bid is accepted with probability at least 1/4.

9. 假设你正在为在大众拍卖网站上销售物品设计策略. 与其他的拍卖网站不同，这个网站使用一次性拍卖，每一次报价必须立即(且不可撤回的)被接受或拒绝. 具体地说，网站如下工作.

- 首先卖家举起一件要卖的东西.
- 然后买家依次上场.
- 当买家 i 上场时，他或她给一个报价 $b_i > 0$.
- 卖家必须立即决定接受还是拒绝这个报价.
- 如果卖家接受这个报价，那么这件物品售出，后面所有的买家都走开. 如果卖家拒绝这个报价，那么买家 i 离开且这个报价被撤销，而且只有在这时卖家才能看见下一个买家.

假设有一件物品提供拍卖，并有 n 个买家，每一个买家报的价钱都不相同. 又假设买家以随机的顺序上场并且卖家知道买家的人数 n. 我们可能希望设计一种策略使得卖家有合理的机会接受 n 个报价中的最高价. 所谓策略是一种规则，卖家按照这个规则只根据 n 的值和到现在为止看到的报价决定是否接受每一个提交的报价.

例如，卖家可以总是接受第一个提交的报价. 其结果是卖家仅以 $1/n$ 的概率接受 n 个报价中的最高价，因为它要求最高报价是第一个提交的.

给出一个策略，要求按照这个策略卖家以不小于 1/4 的概率接受 n 个报价中的最高价，而与 n 的值无关(为了简单起见，你可以假设 n 是偶数). 证明你的策略达到这个概率保证.

[1] ex437.89.251

Let X be a random variable equal to the number of times that b^* is updated. We write $X = X_1 + X_2 + \cdots + X_n$, where $X_i = 1$ if the i^{th} bid in order causes b^* to be updated, and $X_i = 0$ otherwise.

So $X_i = 1$ if and only if, focusing just on the sequence of the first i bids, the largest one comes at the end. But the largest value among the first i bids is equally likely to be anywhere, and hence $EX_i = 1/i$.

Alternately, the number of permutations in which the number at position i is larger than any of the numbers before it can be computed as follows. We can choose the first i numbers in $\binom{n}{i}$ ways, put the largest in position i, order the remainder in $(i-1)!$ ways, and order the subsequent $(n-i)$ numbers in $(n-i)!$ ways. Multiplying this together, we have $\binom{n}{i}(i-1)!(n-i)! = n!/i$. Dividing by $n!$, we get $EX_i = 1/i$.

Now, by linearity of expectation, we have $EX = \sum_{i=1}^{n} EX_i = \sum_{i=1}^{n} 1/i = H_n = \Theta(\log n)$.

10. 考虑一个非常简单的如下工作的在线拍卖系统. 有 n 个报价代理，代理 i 有一个报价 b_i，它是一个正的自然数. 我们假设所有的报价 b_i 彼此不同. 报价代理按照等可能随机选择的顺序上场，轮流地每一个提交自己的报价 b_i，系统始终维护一个变量 b^*，它等于到当前为止看到的最高报价. (开始时令 b^* 为 0.)

作为问题中参数的函数，在这个过程的执行中 b^* 被更新的期望次数是多少？

例 设 $b_1 = 20, b_2 = 25, b_3 = 10$，报价人抵达的顺序是 1,3,2. 那么，对 1 和 2 更新 b^*，而对 3 不更新 b^*.

[1] ex547.67.324

(a) Let's look at a given machine p. In order for it to have no job, every job must be assigned to a different machine. As the jobs are assigned randomly and uniformly, the probability that a given job j is not assigned to p is $(1-\frac{1}{k})$ and therefore the probability that p doesn't get any job is $(1-\frac{1}{k})^k$. Therefore the expected number of machines with no jobs is $N(k) = k(1-\frac{1}{k})^k$.

Finally $N(k)/k = (1-\frac{1}{k})^k$, which goes to $1/e$ as k goes to infinity. Also notice that in the limit the number of machines with no jobs is k/e.

(b) There is a very simple solution to this problem. We notice that the number of rejected jobs (denote it by N_{rej}) is the number of total jobs k minus the number of accepted jobs N_{acc} ($N_{rej} = k - N_{acc}$). The number of jobs accepted is the k minus the number of machines with no jobs N_{nojob} (since the rest of the people do exactly 1 job). Therefore $N_{rej} = k - N_{acc} = k - (k - N_{nojob}) = N_{nojob}$. Therefore the answer to part **(b)** is the same as the answer to part **(a)**.

(c) This part will involve slight calculations. We know that the number of machines with no jobs is k/e (from the first part). We first calculate the number of machines with exactly one job. Again look at a machine p. The probability that only 1 job is assigned to that machine is $k\frac{1}{k}(1-\frac{1}{k})^{k-1}$. (The chance of a given job j being assigned to p is $1/k$ and the probability that the remaining jobs will not be assigned to p is $(1-\frac{1}{k})^{k-1}$. Finally there are k choices of the "given" job j which puts the coefficient k in the beginning). Notice that this also in the limit $1/e$ therefore the number of machines with exactly 1 jobs is also k/e.

Finally the remaining machines regardless of how many jobs they were assigned will perform exactly two jobs. There are $k - \frac{2k}{e}$ of these.

The final tally is k/e machines with one job and $k - \frac{2k}{e}$ people with two jobs. Subtracting this from k (the total number of jobs) we get that $\frac{k(3-e)}{e}$ jobs are rejected, which is approximately 11%.

11. 并行和分布式系统的负载均衡算法试图把一组计算任务分散到多台机器上. 用这种方式,没有一台机器成为"热点". 如果能够进行某种集中的协调,那么负载可能被分散得近乎理想. 如果任务来自各种不能协调的来源,那怎么办? 正如我们在 13.10 节中看到的,一种可能的做法是把它们随机地分配给机器,并希望这种随机化能防止不均衡. 显然,一般地这不能做得像集中解决那样理想,但它可能是相当有效的. 现在我们打算分析 13.10 节中考虑的简单的负载均衡启发式算法的几个变形与推广.

假设有 k 台机器和 k 项要处理的任务. 每一项任务被独立地随机分配给一台机器(每一台机器的可能性相等).

(a) 设 $N(k)$表示没有接受到任务的机器的期望数,因而 $N(k)/k$ 是无事可做的机器的期望比例. 极限 $\lim_{k\to\infty} N(k)/k$是多少? 证明你的答案.

(b) 假设机器不能让剩余的任务排队等候,因而随机分配把一件以上的任务送给机器 M,那么 M 将做它接受到的第一项任务并拒绝其余的任务. 设 $R(k)$是被拒绝的任务的期望数,因而 $R(k)/k$ 是被拒绝的任务的期望比例. 极限 $\lim_{k\to\infty} R(k)/k$ 是多少? 证明你的答案.

(c) 现在假设机器有稍微大一点的缓冲器,每一台机器能做它接受的前两项任务并拒绝其他更多的任务. 设 $R_2(k)$表示在这个规则下被拒绝的任务的期望数. 极限 $\lim_{k\to\infty} R_2(k)/k$ 是多少? 证明你的答案.

Consider a graph G with nodes s and t, and $n-2$ other nodes $v_1, \ldots, v_{n-2}$. There are two parallel edges from s to each v_i, and one edge from v_i to t. The minimum s-t cut is to separate t by itself.

If we run the version of the contraction algorithm described in the problem, it will independently contract each of the length-2 paths from s to t in some order. In order for it to find the minimum s-t cut, it must contract each v_i into s, not into t. There is a $2/3$ chance of this happening for each i, so the probability that the minimum s-t cut is found is $(2/3)^{n-2}$, an exponentially small quantity.

(Note that this example poses no problem for the global minimum cut, which consists of any of the nodes v_i on its own.)

12. 考虑下述求最小 s-t 割的算法，它类似于 Karger 算法. 我们将迭代地用下述随机过程收缩边. 在给定的迭代中，设 s 和 t 分别表示包含原始结点 s 和 t 的可能已被收缩的结点. 为了保证 s 和 t 不被收缩成一点，在每一次迭代时删除连接 s 和 t 的所有边并从剩余的边中选择一条随机的边收缩. 给出一个例子说明这个方法找到最小 s-t 割的概率可能指数地小.

[1] ex242.186.32

The mean for X_2 is n, so in order to have $X_1 - X_2 > c\sqrt{n}$, we need $X_2 < E[X_2] - \frac{c}{2}\sqrt{n} = (1-\delta)E[X_2]$ for $\delta = \frac{c}{2\sqrt{n}}$. Plugging this into the Chernoff lower bound, the probability this happens is

$$e_{-\frac{1}{2}\delta^2 E[X_2]} = e^{-c^2/4}.$$

This can be made smaller than a constant ε by choosing the undetermined constant c large enough.

13. 考虑有 $2n$ 个小球和只有 2 只箱子的小球与箱子的试验. 和通常一样，每一个小球独立地选择 2 只箱子中的一只，两只箱子的可能性相等. 每一只箱子中小球的期望数是 n. 在这个问题中，我们研究它们的差可能大到什么程度的问题. 设 X_1 和 X_2 分别表示这两只箱子中的小球数. (X_1 和 X_2 是随机变量.) 试证明：对任意的 $\varepsilon > 0$，存在一个常数 $c > 0$ 使得概率 $\Pr[X_1 - X_2 > c\sqrt{n}] \leqslant \varepsilon$.

[1] ex646.944.578

14. 某人正在设计并行的物理模拟，带着下述问题来找你. 他们有 k 个基本过程的集合 P，要把每一个过程分配在两台机器 M_1 和 M_2 中的一台上运行. 然后他们要运行 n 项任务 $J_1,\cdots,J_n$ 组成的序列. 每一项任务 J_i 用恰好有 $2n$ 个基本过程的集合 $P_i\subseteq P$ 表示，当这项任务被执行时，这些过程必须正在(它们各自被分配的机器上)运行. 如果对每一项任务 J_i，恰好给这两台机器中的每一台分配 n 个与 J_i 相关联的基本过程，则称这个基本过程对机器的分配是完美均衡的. 如果对每一项任务 J_i，没有多于 $\frac{4}{3}n$ 个与 J_i 相关联的基本过程被分配给同一台机器，则称这个基本过程对机器的分配是接近均衡的.

(a) 证明对任意大的 n，存在任务序列 $J_1,\cdots,J_n$ 使得对这个序列不存在完美均衡分配.

(b) 设 $n\geqslant 200$. 试给一个算法，输入任意的任务序列 $J_1,\cdots,J_n$，产生一个接近均衡的基本过程对机器的分配. 你的算法可以是随机的，这时它的期望运行时间必须是多项式的并且必须永远产生正确的答案.

(a) Let n be odd, $k = n^2$, and represent the set of basic processes as the disjoint union of n sets $X_1, \ldots, X_n$ of cardinality n each. The set of processes P_i associated with job J_i will be equal to $X_i \cup X_{i+1}$, addition taken modulo n.

We claim there is no perfectly balanced assignment of processes to machines. For suppose there were, and let Δ_i denote the number of processes in X_i assigned to machine M_1 minus the number of processes in X_i assigned to machine M_2. By the perfect balance property, we have $\Delta_{i+1} = -\Delta_i$ for each i; applying these equalities transitively, we obtain $\Delta_i = -\Delta_i$, and hence $\Delta_i = 0$, for each i. But this is not possible since n is odd.

(b) Consider independently assigning each process i a *label* L_i equal to either 0 or 1, chosen uniformly at random. Thus we may view the label L_i as a 0-1 random variable. Now for any job J_i, we assign each process in P_i to machine M_1 if its label is 0, and machine M_2 if its label is 1.

Consider the event E_i, that more than $\frac{4}{3}n$ of the processes associated with J_i end up on the same machine. The assignment will be nearly balanced if none of the E_i happen. E_i is precisely the event that $\sum_{t \in J_i} L_t$ either exceeds $\frac{4}{3}$ times its mean (equal to n), or that it falls below $\frac{2}{3}$ times its mean. Thus, we may upper-bound the probability of E_i as follows.

$$\begin{aligned}
\Pr[E_i] &\leq \Pr[\sum_{t \in J_i} L_t < \frac{2}{3}n] + \Pr[\sum_{t \in J_i} L_t > \frac{4}{3}n] \\
&\leq \left(e^{-\frac{1}{2}(\frac{1}{3})^2}\right)^n + \left(\frac{e^{\frac{1}{3}}}{\left(\frac{4}{3}\right)^{\frac{4}{3}}}\right)^n \\
&\leq 2 \cdot .96^n.
\end{aligned}$$

Thus, by the union bound, the probability that any of the events E_i happens is at most $2n \cdot .96^n$, which is at most .06 for $n \geq 200$.

Thus, our randomized algorithm is as follows. We perform a random allocation of each process to a machine as above, check if the resulting assignment is perfectly balanced, and repeat this process if it isn't. Each iteration takes polynomial time, and the expected number of iterations is simply the expected waiting time for an event of probability $1 - .06 - .94$, which is $1/.94 < 2$. Thus the expected running time is polynomial.

This analysis also proves the *existence* of a nearly balanced allocation for any set of jobs.

(Note that the algorithm can run forever, with probability 0. This doesn't cause a problem for the expectation, but we can deterministically guarantee termination without hurting the running time very much as follows. We first run k iterations of the randomized algorithm; if it still hasn't halted, we now find the nearly balanced assignment that is guaranteed to exist by trying all 2^k possible allocations of processes to machines, in time $O(n^2 \cdot 2^k)$. Since this brute-force step occurs with probability at most $.06^k$, it adds at most $O(n^2 \cdot .12^k) = O(n^2 \cdot .12^n) = o(1)$ to the expected running time.)

[1] ex41.971.873

We imagine dividing the set S into 20 *quantiles* $Q_1, \ldots, Q_{20}$, where Q_i consists of all elements that have at least $.05(i-1)n$ elements less than them, and at least $.05(20-i)n$ elements greater than them. Choosing the sample S' is like throwing a set of numbers at random into bins labeled with $Q_1, \ldots, Q_{20}$.

Suppose we choose $|S'| = 40,000$ and sample with replacement. Consider the event $\mathcal{E}$ that $|S' \cap Q_i|$ is between 1800 and 2200 for each i. If $\mathcal{E}$ occurs, then the first nine quantiles contain at most 19,800 elements of S', and the last nine quantiles do as well. Hence the median of S' will belong to $Q_{10} \cup Q_{11}$, and thus will be a (.05)-approximate median of S.

The probability that a given Q_i contains more than 2200 elements can be computed using the Chernoff bound (4.1), with $\mu = 2000$ and $\delta = .1$; it is less than

$$\left[\frac{e^{.05}}{(1.05)^{(1.05)}}\right]^{10000} < .0001.$$

The probability that a given Q_i contains fewer than 1800 elements can be computed using the Chernoff bound (4.2), with $\mu = 2000$ and $\delta = .1$; it is less than

$$e^{-(.5)(.1)(.1)2000} < .0001.$$

Applying the Union Bound over the 20 choices of i, the probability that $\mathcal{E}$ does not occur is at most $(40)(.0001) = .004 < .01$.

15. 假设给你一个非常大的实数集合 S，你想通过抽样给出这些数的中值的近似. 你可以假设 S 中所有的数都是不同的. 令 $n=|S|$，如果 S 中至少有 $\left(\frac{1}{2}-\varepsilon\right)n$ 个数小于 x 并且至少有 $\left(\frac{1}{2}-\varepsilon\right)n$ 个数大于 x，则称 x 是 S 的 ε-近似中值.

考虑如下运行的算法. 等可能地随机选择一个子集 $S' \subseteq S$，计算 S' 的中值，并返回这个值作为 S 的近似中值. 试证明存在一个独立于 n 的绝对常数 c，使得如果样本 S' 的大小为 c，则被返回的数以不少于 0.99 的概率为 S 的 0.05-近似中值（你可以考虑用带放回的抽样构造 S' 的算法，使得 S 的每一个元素可能被选中多次；也可以考虑用不放回的抽样构造 S' 的算法）.

[1] ex835.763.619

One algorithm is the following.

```
For i = 1, 2, ..., n
  Receiver j computes β_ij = f(β*_1 ··· β*_{i-1}, α_i^{⟨j⟩}).
  β*_i is set to the majority value of β_ij, for j = 1, ..., k.
  End for
Output β*
```

We'll make sure to choose an odd value of k to prevent ties.

Let $X_{ij} = 1$ if $\alpha_i^{\langle j \rangle}$ was corrupted, and 0 otherwise. If a majority of the bits in $\{\alpha_i^{\langle j \rangle} : j = 1, 2, \ldots, k\}$ are corrupted, then $X_i = \sum_j X_{ij} > k/2$. Now, since each bit is corrupted with probability $\frac{1}{4}$, $\mu = \sum_j EX_{ij} = k/4$. Thus, by the Chernoff bound, we have

$$\begin{aligned} \Pr[X_i > k/2] &= \Pr[X_i > 2\mu] \\ &< \left(\frac{e}{4}\right)^{k/4} \\ &\leq (.91)^k. \end{aligned}$$

Now, if

$$k \geq 11 \ln n > \frac{\ln n - \ln .1}{\ln(1/.91)},$$

then

$$\Pr[X_i > k/2] < .1/n.$$

(So it is enough to choose k to be the smallest odd integer greater than $11 \ln n$.) Thus, by the union bound, the probability that *any* of the sets $\{\alpha_i^{\langle j \rangle} : j - 1, 2, \ldots, k\}$ have a majority of corruptions is at most .1.

Assuming that a majority of the bits in each of these sets are not corrupted, which happens with probability at least .9, one can prove by induction on i that all the bits in the reconstructed message β^* will be correct.

16. 考虑下述在发送人与接收人之间秘密传输消息的(部分详细规定的)方法. 一个消息表示成一个位串. 令 $\Sigma = \{0,1\}$, Σ^* 表示所有 0 位或多位的位串(例如, 0, 00, 1110001 $\in \Sigma^*$). 0 位的"空串"记作 $\lambda \in \Sigma^*$.

发送人和接收人共享一个秘密函数 $f: \Sigma^* \times \Sigma \to \Sigma$. 也就是说, f 取一个位串和一个二进制位, 返回一个二进制位. 当接收人接收到一个位序列 $\alpha \in \Sigma^*$ 时, 他或她用下述方法译解它.

设 $\alpha = \alpha_1\alpha_2\cdots\alpha_n$, 其中 n 是 α 中的位数
目标是产生一个 n 位译解出的消息 $\beta = \beta_1\beta_2\cdots\beta_n$
令 $\beta_1 = f(\lambda, \alpha_1)$
For $i = 2, 3, \cdots, n$
  令 $\beta_i = f(\beta_1\beta_2\cdots\beta_{i-1}, \alpha_i)$
Endfor
Output β

可以把这看做一种"带反馈的流密码". 这个方法的一个问题是, 如果任何一位 α_i 在传输中发生错误, 那么对所有的 $j \geqslant i$ 它将讹用计算出来的 β_j 值.

我们考虑下述问题. 发送人 S 要把相同的(明文)消息 β 传输给 k 个接收人 $R_1, \cdots, R_k$. 他与每一个接收人享有不同的秘密函数 $f^{(i)}$. 于是, 他给每一个接收人发送不同的加密消息 $\alpha^{(i)}$, 当用函数 $f^{(i)}$ 运行上述算法时 $\alpha^{(i)}$ 被译解成 β.

不幸的是, 通信通道的噪声很重, 使得每一个传输的每一位独立地以概率 1/4 被传错(即, 翻转成它的补). 于是, 可能没有一位接收人自己能够正确地译解这条消息. 试证明如果 k 作为 n 的函数充分的大, 则 k 位接收人能够联合起来用下述方式重新构造明文消息. 他们集合在一起, 不展示任何 $\alpha^{(i)}$ 和 $f^{(i)}$, 交互式地运行一个算法以不少于 9/10 的概率产生正确的 β. (在你的算法中 k 必须多大?)

Let Y denote the number of steps in which your net profit is positive. Then $Y = Y_1 + Y_2 + \cdots + Y_n$, where $Y_k = 1$ if your net profit is positive at step k, and 0 otherwise.

Now, consider a particular step k. $Y_k = 1$ if and only if you have had more than $k/2$ steps in which your profit increased. Since the expected number of steps in which your profit increased is $k/3$, we can apply the Chernoff bound (4.1) with $\mu = k/3$ and $1 + \delta = 3/2$ to conclude that EY_k is bounded by

$$\left[\frac{e^{1/2}}{(3/2)^{(3/2)}}\right]^{(k/3)} < (.97)^k.$$

Thus,

$$EY = \sum_{k=1}^{n} EY_k < \sum_{k=1}^{n} (.97)^k < \frac{1}{1-(.97)} < 34,$$

which is a constant independent of n.

17. 考虑下述简单的赔多赢少的赌博模型. 开始时你的净赢额为 0. 你玩 n 轮，在每一轮你的净赢额以概率 1/3 增加 1，以概率 2/3 减少 1.

证明你的净赢额为正的期望轮数不超过一个与 n 值无关的绝对常数.

[1] ex251.139.906

18. 在这个问题中你考虑下述**顶点覆盖问题**的简单随机算法.

开始时 $S=\emptyset$

While S 不是一个顶点覆盖

  选择一条没有被 S 覆盖的边 e

  随机地选择 e 的一个端点(每一个端点的可能性相等)

  把选中的结点加入 S

Endwhile

我们感兴趣的是这个算法选择的顶点覆盖的期望代价.

(a) 这个算法是**最小权顶点覆盖问题**的 c-近似算法吗? 其中 c 是某个常数. 证明你的答案.

(b) 这个算法是**最小规模顶点覆盖问题**的 c-近似算法吗? 其中 c 是某个常数. 证明你的答案.

(提示: 用 p_e 表示在这个算法中边 e 作为未覆盖的边被选中的概率. 你能够用这些概率表示解的期望值吗? 为了用这些概率给出最优值的上界,尝试给出所有与给定结点 v 关联的边的概率之和的上界,即 $\sum_{e\text{与}v\text{关联}} p_e$ 的上界.)

(a) False. A bad example can consist of a single edge $e = (u, v)$. Assume the cost of u is 1 while the cost of v is more than $2c$. The minimum cost of a vertex cover is 1, while the algorithm selects node v with probability 1/2, and hence has expected cost more than c. Alternately we could have u at most 0 and v at most 1. Now the algorithm's expected cost is 1/2, while the optimum is 0.

(b) This is true. Let p_e be the probability that edge e is selected by the algorithm. Note that the algorithm, as given by the problem set, does not specify the selection rule of edges. You may select uncovered edges at random, or by smallest index, etc. The probability p_e will of course depend on what selection rule was used. But any selection rule gives rise to such probabilities. Now we need to notice two facts. First that $\sum_{e \in E} p_c$ is exactly, the expected number of nodes selected by the algorithm. This is true, as every time we select an edge e we add one node to the vertex cover.

Next we consider the sum of the probabilities p_c for edges adjacent to a vertex v. Let $\delta(v)$ denote the set of edges adjacent to vertex v, and consider $\sum_{e \in \delta(v)} p_e$. Note that this is exactly the expected number of edges selected that are adjacent to node v. Let $S(v)$ be the random variable indicating the selected edges adjacent to v. We have that $Exp(|S(v)|) = \sum_{e \in \delta(v)} p_e$. We claim that this expectation is at most 2. This is true as each time an edge in $\delta(v)$ is selected, with 1/2 probability, we use node v cover edge e, and then all edges in $\delta(v)$ are covered, and no more edges in this set will be selected selected. To make this argument precise, let E_i denote the event that at least i edges are selected adjacent to i. Now we have the following inequality for the expected number of edges selected.

$$Exp(|S(v)|) = \sum_i iProb(E_i - E_{i+1}) = \sum_i Prob(E_i) \leq 1 + \sum_{i>1} 2^{i-1} \leq 2,$$

where the inequality $Prob(E_i) \leq 2^{i-1}$ follows for $I > 1$ as after each edge selected adjacent to v we add v to the vertex cover with probability 1/2.

Now we are ready to bound the expected size of the vertex cover compared to the optimum. Let S^* be an optimum vertex cover.

$$\sum_e p_e \leq \sum_{v \in S^*} \sum_{e \in \delta(v)} p_e \leq \sum_{v \in S^*} 2 = 2|S^*|,$$

where the first inequality follows as S^* is a vertex cover, and so the second sum must cover each edge e.

[1] ex593.991.129