問 1

・プログラムの実行方法

$ python q1.py < q1_in.txt > q1_out.txt

・問題の解法説明

考えられる行列を全て列挙した上で、それらの行列式を計算する。その行列式がそれぞれ d に一致するかを確かめ、一致する行列をプログラム(q1.py)で数え上げれば良い。A = {x, y} として、M(3, 3; A)から{9bit で表せる 2 進数全体}には自明な全単射があるので、今回列挙したい行列は 0 以上 2^9 未満の整数で列挙できる。後は det: M(3, 3; A) → Z を実装すれば良い。q1.py では関数 det3 として行列式を実装した。det3 では行列の 0 行目に注目して余因子展開をしている。

・プログラムの正しさの検証

det3 の実装にバグが無いか不安だったので、test_det.py でバグの有無を確かめている。具体的には、numpy.linalg.det で計算される行列式と自作の det3 が 1000 個のランダムな 3✕3 行列に対して(殆ど)同じ値を取るかどうかを確かめた。

問 2

・プログラムの実行方法

$ python q2.py < q2_in.txt > q2_out.txt

・問題の解法説明

問題で定義された文字列 Si の文字列としての長さを Li とおく。また、Si に含まれる文字 a, b, c の数をそれぞれ NAi, NBi, NCi と書くことにする。加えて、文字列 Sk の p 文字目から q 文字目に含まれる文字 a, b, c の数を dfs(k, p, q)と書くことにする(つまり dfs(k, p, q)は 3 つの 0 以上の整数から成るタプルだと思って欲しい)。この時、今回の問題では、入力として与えられる k, p, q に対して、この dfs(k, p, q)を逐一求めれば良いことが分かる。k < 4 の時は、問題文より dfs(k, p, q)の値は直ぐに分かる。一方、k >= 4 の時、Sk = S(k-3) + S(k-2) + S(k-1) と書けるので、この分割された 3 つの文字列それぞれに対して、Sk において p 文字目から q 文字目に該当するような文字 a, b, c の数を数えれば良い。この分割された 3 つの文字列それぞれに対する数え上げは再帰によって実現できる。単なる再帰では大幅に計算時間とメモリを消費してしまうので、適切な枝刈りが必要である。上に述べたような枝狩りは、例えば、文字列 Sk の p 文字目から q 文字目が文字列 Sr (r = k-3, k-2, k-1)を包含している場合(包含は Lr を用いて判定できる)、再帰の代わりに、前計算しておいた NAr, NBr, NCr を用いることで実現できる。

・プログラムの正しさの検証
今回の問題を解く上で q2.py ほど効率的ではないが、より単純なプログラムであるところの q2-brute.py との計算結果を q2.py と比較した。入力としては test_gen2.py から出力された test_input2.txt を用いた。これは k = 10 となるような全ての入力パターンを網羅しており、k <= 10 となるような全てのパターンを網羅していることが分かる。この入力に対する q2.py と q2-brute.py の出力 test_output2.txt と test_output_brute2.txt は一致したので、q2.py は正しい出力をしているだろうと考えられる。

問 3
・プログラムの実行方法
$ python q3.py < q3_in.txt > q3_out.txt

・問題の解法説明
大まかな方針としては、次にやって来る人が座る位置の候補を優先度付きキューに常に格納しておき、優先度付きキューの先頭にある椅子がまだ誰にも座られていないなら、その椅子に新しくやって来た人を座らせれば良い。以下では q3.py にそって解法の説明をしていく。used[i]は左から i 番目の椅子が既に誰かに座られていると True になり、そうでない場合は False とする。また、places[i] := (i + 1 番目にやって来る人が左から何番目の椅子に座るか)をデータとして持つように places を構築していく。つまり、最終的に求めたい答えは i が奇数となるような places[i]の和である。優先度付きキュー(ヒープ)h はタプル(v, i, l, r)を要素として持つ。このタプル要素はそれぞれ以下のような値である。i := タプルが表す椅子は左から何番目か、 l := タプルが表す椅子より左にある、かつ既に人が座っているような椅子で最も右にある椅子は左から何番目か、 r := タプルが表す椅子より右にある、かつ既に人が座っているような椅子で最も左にある椅子は左から何番目か、v := (タプルが表す椅子と既に人が座っている椅子との距離の最小値) ✕ (-1) 。但し、l, r が-1 の時は、上の条件に見合う椅子が存在しないことを表す。例えば左端の椅子に対しては、それより左の椅子は存在しないので l = -1 である。この used、h を更新しながら、places を構築していく。上の定義より、次に places に追加する値は used[i]が False であり、かつ h 内で v が最も小さくなるような i であることが分かる。もし、そのような(v, i, l, r)に新しくやって来た人が座るなら、その次にやって来る人が座る場所の候補として、左から[(l + i)/2]番目の椅子と[(i + r)/2]番目の椅子を h に追加する。

・プログラムの正しさの検証
問 2 の時と同様に q3.py より愚直かつ効率の悪いプログラムである q3-brute.py と q3.py の実行結果を比較した。入力として test_gen3.py から test_input3.txt を作った。この入力に対する q3.py と q3-brute.py の出力は test_output3.txt と test_output_brute3.txt で同じ。