```
/** \author Anne-Soline GUILBERT--LY
 *  \date   dec 2019
 */

#include <iostream>
#include <vector>
#include <random>
#include <ctime>
#include <cmath>
#include <iomanip>
#include <algorithm>

#define MAX ERROR 0.01
#define MIN DELTA P 0.000001

using namespace std;


constexpr int indice premiere ligne(0);
constexpr double minimum(0.);
constexpr double maximum(1.);

typedef vector<vector<bool>> Grille;

struct Graphe {
    vector<double> p;
    vector<double> pprime;
};


void initialisation libre(Grille& libre, const int& n);

void construire passage(Grille& passage,const Grille& libre,
                        const int& n);

void chercher chemin(Grille& passage, const Grille& libre, const int& n,
                     int i, int j);

void affiche terrain(const Grille& terrain, const int& n,
                     const string& format);

void init libre hasard(Grille& libre, const int& n,
                       default random engine& e,
                       bernoulli distribution& b);

void construire un seul passage(Grille& passage, const Grille& libre,
                                const int& n);

bool terrain traversable(const Grille& terrain, const int& n);

double calcul pprime(Grille& libre, const int& n, Grille& passage,
                     default random engine& e, const double& p,
                     const double& nbterrains);

void init passage a false(Grille& passage, const int& n);

void dichotomie(double& min,double& max,double& error,double& maxprime,
                double& minprime, double& p, double& pprime,
                const int& n,Grille& libre, Grille& passage,
                default random engine& e, const double& nbterrains,
                Graphe& courbe);

void afficher p et pprime(const Graphe& courbe);

void graphe p(Grille& libre,const int& n,Grille& passage,
              default random engine& e, double& p,
              const double& nbterrains);
```

```
 71  int main(){
 72
 73      default random engine e(time(0));
 74
 75      char niveau; cin >>niveau;
 76      int n; cin >> n;
 77
 78      Grille libre(n, vector<bool>(n));
 79      Grille passage(n, vector<bool>(n));
 80
 81      switch (niveau){
 82          case 'a':{
 83              string format("P1");
 84              initialisation libre(libre, n);
 85              construire passage(passage, libre, n);
 86              affiche terrain(passage, n, format);
 87              return 0;
 88          }
 89          case 'b' : {
 90              double p; cin >> p;
 91              int nbt; cin >>nbt;
 92              double nbterrains(nbt);
 93
 94              cout <<setprecision(6)<<fixed;
 95              cout<<calcul pprime(libre, n, passage, e, p, nbterrains);
 96              return 0;
 97          }
 98          case 'c' : {
 99              int nbt; cin >>nbt;
100              double nbterrains(nbt), min(minimum), max(maximum),
101              maxprime(maximum), minprime(minimum), pprime, error, p;
102
103              Graphe courbe={{minimum, maximum}, {minimum, maximum}};
104
105              dichotomie(min, max, error, maxprime, minprime, p, pprime,
106              n, libre, passage, e, nbterrains, courbe);
107              return 0;
108          }
109      }
110  }
111
112
113  void initialisation libre(Grille& libre, const int& n){
114
115      int valeur;
116
117      for(int i(0); i<n; ++i){
118          for(int j(0); j<n; ++j){
119              cin >> valeur;
120              if(valeur == 0){
121                  libre[i][j] = true;
122              }
123          }
124      }
125  }
126
127  void construire passage(Grille& passage,const Grille& libre,
128  const int& n){
129
130      int i;
131
132      for(int j(0); j<n; ++j){
133          i=indice premiere ligne;
134          if(libre[i][j]){
135              passage[i][j]=true;
136              chercher chemin(passage, libre, n, i, j);
137          }
138      }
139  }
140
```

```
141  void chercher chemin(Grille& passage, const Grille& libre, const int& n,
142  int i, int j){
143
144      if((i+1)<=(n-1) and libre[i+1][j] and !passage[i+1][j]){
145          passage[i+1][j]=true;
146          chercher chemin(passage, libre, n, (i+1), j);
147      }
148
149      if((i-1)>=0 and libre[i-1][j] and !passage[i-1][j]){
150          passage[i-1][j]=true;
151          chercher chemin(passage, libre, n, (i-1), j);
152      }
153
154      if((j-1)>=0 and libre[i][j-1] and !passage[i][j-1]){
155          passage[i][j-1]=true;
156          chercher chemin(passage, libre, n, i, (j-1));
157      }
158
159      if((j+1)<=(n-1) and libre[i][j+1] and !passage[i][j+1]){
160          passage[i][j+1]=true;
161          chercher chemin(passage, libre, n, i, (j+1));
162      }
163  }
164
165  void affiche terrain(const Grille& terrain, const int& n,
166  const string& format){
167
168      cout << format <<endl;
169      cout << n <<" "<<n<<endl;
170
171      for(int i(0); i<n; ++i){
172          for(int j(0); j<n; ++j){
173              if(terrain[i][j]){
174                  cout <<"0 ";
175              }else{
176                  cout <<"1 ";
177              }
178              if(j==34){
179                  cout<<endl;
180              }
181          }
182          cout<<endl;
183      }
184  }
185
186  void init libre hasard(Grille& libre,const int& n,
187  default random engine& e, bernoulli distribution& b){
188
189      for(int i(0); i<n; ++i){
190          for(int j(0); j<n; ++j){
191              if(b(e) == 1){
192                  libre[i][j] = true;
193              }else {
194                  libre[i][j] = false;
195              }
196          }
197      }
198  }
199
200  void construire un seul passage(Grille& passage, const Grille& libre,
201  const int& n){
202
203      int i;
204
205      for(int j(0); j<n; ++j){
206          i=indice premiere ligne;
207          if(libre[i][j]){
208              passage[i][j]=true;
209              chercher chemin(passage, libre, n, i, j);
210          }
```

```
211             if(terrain traversable(passage, n)){
212                 break;
213             }
214         }
215     }
216
217     bool terrain traversable(const Grille& terrain, const int& n){
218
219         for(int j(0); j<n; ++j){
220             if(terrain[n-1][j]){
221                 return true;
222             }
223         }
224         return false;
225     }
226
227     double calcul pprime(Grille& libre, const int& n, Grille& passage,
228     default random engine& e, const double& p, const double& nbterrains){
229
230         bernoulli distribution b(p);
231         int compteur(0);
232
233         for (double k(1); k<=nbterrains; ++k){
234             init libre hasard(libre, n, e, b);
235             construire un seul passage(passage, libre, n);
236             if(terrain traversable(passage, n)){
237                 ++compteur;
238             }
239             init passage a false(passage, n);
240         }
241
242         return (compteur/nbterrains);
243     }
244
245     void init passage a false(Grille& terrain, const int& n){
246         Grille tmp(n, vector<bool>(n));
247         terrain = tmp;
248     }
249
250     void dichotomie(double& min, double& max,double& error,double& maxprime,
251     double& minprime, double& p, double& pprime, const int& n,Grille& libre,
252     Grille& passage, default random engine& e, const double& nbterrains,
253     Graphe& courbe){
254
255         do{
256             p = (max+min)/2.;
257             pprime = calcul pprime(libre, n, passage, e, p, nbterrains);
258             error = pprime - (maxprime + minprime)/2.;
259
260             courbe.p.push back(p);
261             courbe.pprime.push back(pprime);
262
263             if(error>0.){
264                 max = p;
265                 maxprime = calcul pprime(libre,n,passage,e,max,nbterrains);
266             }
267             if(error<0.){
268                 min = p;
269                 minprime = calcul pprime(libre,n,passage,e,min,nbterrains);
270             }
271
272         }while((max-min)>MIN DELTA P and abs(error)>MAX ERROR);
273
274         sort(courbe.p.begin(), courbe.p.end());
275         sort(courbe.pprime.begin(), courbe.pprime.end());
276         afficher p et pprime(courbe);
277     }
278
279     void afficher p et pprime(const Graphe& courbe){
280
```

```
281         for(unsigned int i(0); i<courbe.p.size(); ++i){
282             cout <<setprecision(6)<<fixed;
283             cout<<courbe.p[i]<<" "<<courbe.pprime[i]<<endl;
284         }
285     }
286
287
288
289
290
291
292
```