```cpp
/*******************************************************************
* Some other definitions                                           *
*******************************************************************/

#define _inline(f...) f() __attribute__((always_inline)); f
#define _foreach(it, b, e) for (typeof(b) it = (b); it != (e); it++)
#define foreach(x...) _foreach(x)
#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()

/*******************************************************************
* Field with barriers                                              *
*******************************************************************/
#include <list>
#include <set>
const int TAM = 2000;
typedef pair<point, point> segment;
typedef pair<int, int> barrier;
struct field {
  int n, m;
  point v[TAM];
  barrier b[TAM];
  list<int> e[TAM];
  field(): n(0), m(0) {}
  void clear() {
     for (int i = 0; i < n; i++) e[i].clear();
     n = m = 0;
  }
  int ccw(int a, int b, int c) { return ::ccw(v[a], v[b], v[c]); }
  void make_barrier(int i, int j) {
     e[i].push_back(m); e[j].push_back(m);
     b[m++] = barrier(i, j);
  }

  //Removes the degenerate cases
  void normalize() {
     set&lt;segment&gt; T; set<point> U;
     for (int i = 0; i < n; i++) make_barrier(i, i);
     for (int i = 0; i < m; i++) {
       point p = v[b[i].first], q = v[b[i].second];
       set<point> S;
       S.insert(p); S.insert(q);
       for (int j = 0; j < m; j++) {
         point r = v[b[j].first], s = v[b[j].second];
         if (r == p || r == q || s == p || s == q) continue;
         if (cmp((q - p) % (s - r)) == 0) {
           if (between(p, r, q)) S.insert(r);
           if (between(p, s, q)) S.insert(s);

          } else if (seg_intersect(p, q, r, s)) {
            S.insert(line_intersect(p, q, r, s));
          }
        }
        foreach (st, all(S)) {
          if (st != S.begin()) T.insert(segment(p, *st));
```

```
          U.insert(p = *st);
        }
      }
      clear();
      foreach (it, all(U)) v[n++] = *it;
      foreach (it, all(T)) {
        int i = lower_bound(v, v+n, it->first) - v;
        int j = lower_bound(v, v+n, it->second) - v;
        make_barrier(i, j);
      }
    }
    //Poggi-Moreira-Fleischman-Cavalcante Algorithm
    //Determine the graph with all edges in an eventual minimum path
    void pmfc(graph& G) {
      int sel[TAM][2], active[TAM];
      for (int i = 0; i < n; i++) {
        vector< pair<double, int> > T;
        foreach (it, all(e[i])) {
          int j = b[*it].first + b[*it].second - i;
          T.push_back(make_pair(arg(v[j] - v[i]), j));
        }
        sort(all(T));
        if (T.empty()) T.push_back(make_pair(0, i));
        active[i] = 0;
        int p = T.size();
        for (int j = 0; j < p; j++) {
          sel[i][0] = T[j].second; sel[i][1] = T[(j+1) % p].second;
          if (ccw(sel[i][0], sel[i][1], i) <= 0) {
            active[i] = 1; break;
          }
        }
      }
      G.init(n);
      for (int i = 0; i < n; i++) for (int j = 0; j < i; j++) {
        if (!active[i] || !active[j]) continue;
        if (ccw(i, j, sel[i][0]) * ccw(i, j, sel[i][1]) == -1 || \
            ccw(i, j, sel[j][0]) * ccw(i, j, sel[j][1]) == -1)
          continue;
        for (int k = 0; k < m; k++) {
          int org = b[k].first, dest = b[k].second;
          if (org == i||org == j||dest == i||dest == j) continue;
          if (seg_intersect(v[i], v[j], v[org], v[dest])) goto PROX;
        }
        G.aresta(i, j, 1, abs(v[j] - v[i]));
PROX: ;
      }
    }
};

/*************************************************************
* Arbitrary precision arithmetics                            *
*************************************************************/
#include <sstream>
const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51 --> BASE = 10^DIG
```

```
const int TAM = 2048;
struct bigint {
  int v[TAM], n;
  bigint(int x = 0): n(1) {
    memset(v, 0, sizeof(v));
    v[n++] = x; fix();
  }
  bigint(char *s): n(1) {
    memset(v, 0, sizeof(v));
    int sign = 1;
    while (*s && !isdigit(*s)) if (*s++ == '-') sign *= -1;
    char *t = strdup(s), *p = t + strlen(t);
    while (p > t) {
      *p = 0; p = max(t, p - DIG);
      sscanf(p, "%d", &v[n]);
      v[n++] *= sign;
    }
    free(t); fix();
  }
  bigint& fix(int m = 0) {
    n = max(m, n);
    int sign = 0;
    for (int i = 1, e = 0; i <= n || e && (n = i); i++) {
      v[i] += e; e = v[i] / BASE; v[i] %= BASE;
      if (v[i]) sign = (v[i] > 0) ? 1 : -1;
    }
    for (int i = n - 1; i > 0; i--)
      if (v[i] * sign < 0) { v[i] += sign * BASE; v[i+1] -= sign; }
    while (n && !v[n]) n--;
    return *this;
  }
  int cmp(const bigint& x = 0) const {
    int i = max(n, x.n), t = 0;
    while (1) if ((t = ::cmp(v[i], x.v[i])) || i-- == 0) return t;
  }
  bool operator <(const bigint& x) const { return cmp(x) < 0; }
  bool operator ==(const bigint& x) const { return cmp(x) == 0; }
  bool operator !=(const bigint& x) const { return cmp(x) != 0; }
  operator string() const {
    ostringstream s; s << v[n];
    for (int i = n - 1; i > 0; i--) {
      s.width(DIG); s.fill('0'); s << abs(v[i]);
    }
    return s.str();
  }
  friend ostream& operator <<(ostream& o, const bigint& x) {
    return o << (string) x;
  }

   bigint& operator +=(const bigint& x) {
     for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
     return fix(x.n);
   }
   bigint operator +(const bigint& x) { return bigint(*this) += x; }
   bigint& operator -=(const bigint& x) {
```

```
    for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
    return fix(x.n);
  }
  bigint operator -(const bigint& x) { return bigint(*this) -= x; }
  bigint operator -() { bigint r = 0; return r -= *this; }
  void ams(const bigint& x, int m, int b) {//*this+=(x * m)<<b;
    for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
      v[i+b] += x.v[i] * m + e; e = v[i+b] / BASE; v[i+b] %= BASE;
    }
  }
  bigint operator *(const bigint& x) const {
    bigint r;
    for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
    return r;
  }
  bigint& operator *=(const bigint& x) {return *this = *this * x;}
  // cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
  bigint div(const bigint& x) {
    if (x == 0) return 0;
    bigint q; q.n = max(n - x.n + 1, 0);
    int d = x.v[x.n] * BASE + x.v[x.n-1];
    for (int i = q.n; i > 0; i--) {
      int j = x.n + i - 1;
      q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
      ams(x, -q.v[i], i-1);
      if (i == 1 || j == 1) break;
      v[j-1] += BASE * v[j]; v[j] = 0;
    }
    fix(x.n); return q.fix();
  }
  bigint& operator /=(const bigint& x) { return *this = div(x); }
  bigint& operator %=(const bigint& x) { div(x); return *this; }
  bigint operator /(const bigint& x) {return bigint(*this).div(x);}
  bigint operator %(const bigint& x) {return bigint(*this) %= x;}
  bigint pow(int x) {
    if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
    bigint r = 1;
    for (int i = 0; i < x; i++) r *= *this;
    return r;
  }
  bigint root(int x) {
    if (cmp() == 0 || cmp() < 0 && x % 2 == 0) return 0;
    if (*this == 1 || x == 1) return *this;
    if (cmp() < 0) return -(-*this).root(x);
    bigint a = 1, d = *this;
    while (d != 1) {
      bigint b = a + (d /= 2);
      if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
    }
    return a;
  }
};
```

```
/**************************************************************************
* BOOKCASE                                                                *
**************************************************************************/

int main () {
  int runs; cin>>runs;
  while (runs--) {
    int N;
    cin>>N;
    vector<pair<int,int> > b(N);
    for (int i=0; i<N; i++)
      cin>>b[i].first>>b[i].second;
    sort(b.begin(),b.end(),greater<pair<int,int> >());

    vector<int> h(N),t(N),tsum(N+1);
    for (int i=0; i<N; i++) {
      h[i]=b[i].first;
      t[i]=b[i].second;
      tsum[i+1]=tsum[i]+t[i];
    }

    vector<vector<int> > H(tsum[N]+1, vector<int>(tsum[N]+1, INF));
    H[0][0]=0;

    for (int i=0; i<N; i++)
      for (int T1=tsum[i+1]; T1>=0; T1--)
        for (int T2=tsum[i+1]; T2>=0; T2--) {
          if (T1> t[i]) H[T1][T2] <?= H[T1-t[i]][T2];
          if (T1==t[i]) H[T1][T2] <?= H[T1-t[i]][T2] + h[i];
          if (T2> t[i]) H[T1][T2] <?= H[T1][T2-t[i]];
          if (T2==t[i]) H[T1][T2] <?= H[T1][T2-t[i]] + h[i];
        }

    int res = INT_MAX;
    for (int T1=1; T1<=tsum[N]; T1++)
      for (int T2=1; T2<=tsum[N]; T2++)
        res <?= (T1 >? T2 >? (tsum[N]-T1-T2)) * (h[0]+H[T1][T2]);
    cout << res << endl;
  }
}

/**************************************************************************
* VASE COLLECTOR                                                          *
**************************************************************************/
long long v[36]; int bitcnt[1<<18];

int countbits(long long a) {return bitcnt[a>>18]+bitcnt[a&0x3FFFF];}

int check(int n, int cur, int sel, long long mask) {
  if (countbits(mask)<n) return 0;
  if (sel==n) return 1;
  if (sel+36-cur<n) return 0;
  return check(n,cur+1,sel,mask) || check(n,cur+1,sel+1,mask&v[cur]);
}
```

```
int main() {
  for(int i=0;i<(1<<18);i++) {
        int cnt=0;
        for(int j=0;j<18;j++)
          if ((1<<j)&i) cnt++;
        bitcnt[i]=cnt;
  }

  int N;
  cin >> N;
  while (N--) {
        int m;
        cin >> m;
        for(int i=0;i<36;i++) v[i]=0;
        for(int i=0;i<m;i++) {
          int x,y;
          cin >> x >> y;
          v[y-1]|=(1LL<<(x-1));
        }

        int n;
        for(n=2;n*n<=m && check(n,0,0,(1LL<<36)-1);n++);
        cout << n-1 << endl;
  }
  return 0;
}

/*********************************************************************
* Transversing a maze (Ro and Bot Meet - ACM 2117)                    *
* Transversing a maze using left or right hand rule. In this         *
* code the maze nxm is represented as an int M[n][m] matrix in       *
* which each cell is a bit mask indicating if the path in some       *
* direction is open or not. We have:                                 *
*   ---------   The one bit in the mask indicates that a direction *
*   |   0   |   is open. Ro transverses the maze using the left    *
*   |3     1|   hand rule and Bot using the right hand rule.       *
*   |   2   |                                                      *
*   ---------                                                      *
*********************************************************************/
#define between(i,a,b) ( (a<=i) && (i<=b) )
#define in_maze(i,j) ((0<=i)&&(i<n)&&(0<=j)&&(j<m))

int getValue( char c ) {
  if ( c >= 'A' && c <= 'F' ) return ( c - 'A' ) + 10;
  else if ( c >= 'a' && c <= 'f' ) return ( c - 'a' ) + 10;
  else return ( c - '0' );
}

int M[50][50]; int diri[4] = {-1,0,1,0}; int dirj[4] = {0,1,0,-1};

int main() {
  int n,m, maze=1;
  char ch[10];
  int roi, roj, rod, boti, botj, botd;
  while (scanf("%d %d", &n, &m), n || m)  {
```

```
    rep(i,n) rep(j,m) {
      scanf("%s", ch);
      M[i][j] = getValue( ch[0] );
    }
    roi = roj = 0; rod = (M[0][0] & (1<<3)) ? 1 : 2;
    boti = n-1; botj = m-1; botd = (M[n-1][m-1] & (1<<1)) ? 3 : 0;
    bool found = false;
    while (in_maze(roi,roj) && in_maze(boti,botj))
    {
      if ((roi == boti) && (roj == botj)) {found = true; break;}
      rod = (rod+3)%4; botd = (botd+1)%4;
      while (!(M[roi][roj] & (1<<rod))) rod = (rod + 1)%4;
      while (!(M[boti][botj] & (1<<botd))) botd = (botd + 3)%4;
      roi += diri[rod]; roj += dirj[rod];
      boti += diri[botd]; botj += dirj[botd];
    }
    printf("Maze %d: The robots ", maze++);
    if (found) printf("meet in row %d, column %d.\n\n", roi+1,
roj+1);
    else printf("do not meet.\n\n");
  }
}

/*************************************************************************
* EURO EFFICIENCE - minimize coins used ( including change )             *
*************************************************************************/
#define between(x,a,b) ( (a<x) && (x<b) )
#define p_rep(i,n) for (int i=1; i<=n; i++)
int main() {
  int testcases, sum, c, s, d[101]; bool finished;
  scanf("%d", &testcases);
  while(testcases--){
    s = 1; finished = false;
    p_rep(i,100) d[i] = INF;
    rep(i,6) { scanf("%d", &c); d[c] = 1; } // coin values
    while (!finished) {
      finished = true; ++s;
      p_rep(i,100) {
        if (d[i] == INF) finished = false;

        p_rep(j,100) if (d[i]+d[j] == s)
        {
          if (i+j <= 100) d[i+j] <?= s;
          if (i-j > 0) d[i-j] <?= s;
          if (j-i > 0) d[j-i] <?= s;
        }
      }
    }

    s = sum = 0;
    p_rep(i,100) {
      s >?= d[i]; sum += d[i];
    }
    printf("%4.2f %d\n", ((float)sum) / 100., s);
  }
```

```
}

/**************************************************************
* STABLE MARRIAGE - match m men to n woman in a stable and monogamic*
* fashion. L[i][ ] is the list of women in order of decreasing      *
* preference of man i and R[j][i] is how attractive man i to woman  *
* j, so, if R[j][i1] > R[j][i2] then woman j prefers man i1 to i2.  *
* L2R[i] contains man i's mate and R2L[j] contains the mate of woman*
* j or -1 if she is unmated. Condition: n>=m. Complexity O(m^2)
*
**************************************************************/
int m, n; int L[MAXM][MAXW], R[MAXW][MAXM];
int L2R[MAXM], R2L[MAXW]; int p[MAXM];

void stableMarriage() {
  static int p[MAXM];//man's preferences
  rep(j,n) R2L[j] = -1; rep(k,128) p[k] = 0;

  rep(i,m) {//for each man
    int man = i;
    while( man >= 0 )
    {
      int wom;
      while( 1 ) {//proposes women
        wom = L[man][p[man]++];
        if( R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]] ) break;
      }
      int hubby = R2L[wom]; //old husband from the woman
      R2L[L2R[man] = wom] = man;
      man = hubby; //remarry dumped husband
    }
  }
}

/**************************************************************
* SET RECONSTRUCTION FROM PAIRWISE SUM: Given a multiset S of       *
* integers, reconstructs the multiset V such that S is the multiset *
* of pairwise sums of V.Returns true if successful and false if     *
* confused.                                                         *
**************************************************************/
#include <set>
bool pairsums( int *ans, multiset< int > &seq ) {
    int N = seq.size();
    if( N < 3 ) return false;
    __typeof( seq.end() ) it = seq.begin();
    int a = *it++, b = *it++, i = 2;

    for( ; i * ( i - 1 ) < 2 * N && it != seq.end(); i++, ++it ) {
        // assume seq[i] = ans[1] + ans[2]
        ans[0] = a + b - *it;
        if( ans[0] & 1 ) continue;
        ans[0] >>= 1;
        // try ans[0] as a possible least element
        multiset< int > seq2 = seq;
        int j = 1;
```

```
        while( seq2.size() ) {
            ans[j] = *seq2.begin() - ans[0];
            for( int k = 0; k < j; k++ ) {
                __typeof( seq2.end() ) jt=seq2.find(ans[k]+ans[j]);
                if( jt == seq2.end() ) goto HERE;
                seq2.erase( jt );
            }
            j++;
        }
        HERE:;
        if( j * ( j - 1 ) < 2 * N ) continue;
        // it worked! [modify this to deal with multiple answers]
        return true;
    }
    return false;
}

/**************************************************************************
* POWER MOD: Computes b^p mod m. Wants b >= 0, p >= 0, m >= 1.            *
**************************************************************************/
int powmod( long long b, int p, int m )
{
    long long r = 1;
    for( int i = ( 1 << 30 ); i; i >>= 1 )
    //for( long long i = ( 1LL << 62 ); i; i >>= 1 )//for big powers
    {
        r *= r; r %= m;
        if( p & i ) { r *= b; r %= m; }
    }
    return ( int )r;
}

/**************************************************************************
* ROMAN NUMERALS                                                          *
**************************************************************************/

string fill( char c, int n ) { string s; while( n-- ) s += c; return
s; }

string toRoman( int n ) {
 if( n < 4 ) return fill( 'i', n );
 if( n < 6 ) return fill( 'i', 5 - n ) + "v";
 if( n < 9 ) return string( "v" ) + fill( 'i', n - 5 );
 if( n < 11 ) return fill( 'i', 10 - n ) + "x";
 if( n < 40 ) return fill( 'x', n / 10 ) + toRoman( n % 10 );
 if( n < 60 ) return fill( 'x', 5 - n / 10 ) + 'l' + toRoman( n % 10
);
 if( n < 90 ) return string( "l" ) + fill( 'x', n / 10 - 5 ) +
toRoman( n % 10 );
 if( n < 110 ) return fill( 'x', 10 - n / 10 ) + "c" + toRoman( n %
10 );
 if( n < 400 ) return fill( 'c', n / 100 ) + toRoman( n % 100 );
 if( n < 600 ) return fill( 'c', 5 - n / 100 ) + 'd' + toRoman( n %
100 );
 if( n < 900 ) return string( "d" ) + fill( 'c', n / 100 - 5 ) +
```

```
toRoman( n % 100 );
 if( n < 1100 ) return fill( 'c', 10 - n / 100 ) + "m" + toRoman( n
% 100 );
 if( n < 4000 ) return fill( 'm', n / 1000 ) + toRoman( n % 1000 );
 return "?";
}


/*****************************************************************
* Hungarian Algorithm                                            *
*****************************************************************/

#define SIZE (2*MAX+1)

int mat[SIZE][SIZE];    /* Weighted Matrix */
int matchedWith[SIZE];
int level[SIZE]; int parent[SIZE]; int N;
int weight[SIZE];   /* Weight/Label of each Node */
int S[SIZE];  /* Set of vertices of X/U in Hungarian Tree */
int T[SIZE];  /* Vertices in Y/V and in the current Matching */

void init(void) { rep(i,SIZE) memset(mat[i],0,sizeof(int)*SIZE); }

int augmentPath(int u) {
  int v,w,last; int q[SIZE+1],head,tail;
  memset(level,0,sizeof(int)*(2*N+1));
  memset(parent,0,sizeof(int)*(2*N+1));
  head   = 0;   tail   = 1;  q[1]   = u;  level[u]= 1; last  = 0;
  while((head!=tail) && !last) {
    head++; v = q[head];
    if(level[v]%2) {
      for(w=N+1;w<=2*N;w++) {
        if(!level[w] && (mat[v][w]==weight[v]+weight[w])) {
          if(!matchedWith[w]) {
            parent[w] = v;
            last      = w;
            break;
          }
          else if(matchedWith[w]!=v)/* Unmatched Edge(v,w) */
          {
            parent[w] = v; level[w]  = level[v]+1; q[++tail] = w;
          }
        }
      }
    }
    else
    {
      w        = matchedWith[v];/* Matching partner of v */
      parent[w]  = v;
      q[++tail]  = w;      /* Enque matched Edge (v,w) */
      level[w]  = level[v]+1;
    }
  }

  if(last)  {
```

```
    for(w=last; w; w=parent[parent[w]]) {
      matchedWith[w]     = parent[w];
      matchedWith[parent[w]]  = w;
    }
    return 1;
  }
  else
  {
    for(v=1;v<=N;v++)
    {
      if(level[v])  S[v]   = 1;
      if(level[N+v]) T[N+v]   = 1;
    }
    return 0;
  }
}

void optimalMatching(void) {
  int i,j,max,slack,u,v;
  int nMatch,eps;
  memset(matchedWith,0,sizeof(int)*(2*N+1));
  for(i=1;i<=N;i++) {
    max = 0;
    for(j=N+1;j<=2*N;j++) if(mat[i][j] > max) max = mat[i][j];
    weight[i]   = max; weight[i+N] = 0;
  }
  nMatch   = 0;
  while(1) {
    memset(S,0,sizeof(int)*(2*N+1));
    memset(T,0,sizeof(int)*(2*N+1));
    for(i=1;i<=N;i++) {
      if(!matchedWith[i]) {
        if(augmentPath(i)) nMatch++;
      }
    }
    if(nMatch == N) break;
    eps = INF;
    for(i=1;i<=N;i++) if(S[i]) for(j=N+1;j<=2*N;j++) if(!T[j]){
      slack = weight[i]+weight[j]-mat[i][j];
      if(slack < eps) eps = slack;
    }
    for(i=1;i<=N;i++) {
      if(S[i]) weight[i]    -= eps;
      if(T[i+N]) weight[i+N]  += eps;
    }
  }
  return ;
}

void main(void)

//usage
N =...; init();
for(i=1;i<=N;i++) for(j=N+1;j<=2*N;j++)
        scanf("%d",&mat[i][j]);
```

```
optimalMatching();
answer in matchedWith[i] i=1...N

/*******************************************************************
* Perimeter of the union of rectangles                             *
*******************************************************************/

double union_perimeter(vector<rect>& R) {
  int n = R.size(); if (n == 0) return 0;
  vector< pair<double, int> > E;
  int m = 0;
  for (int i = 0; i < n; i++) {
    E.push_back(make_pair(R[i].x1, i));
    E.push_back(make_pair(R[i].x2, ~i));
    y[m++] = R[i].y1;
    y[m++] = R[i].y2;
  }
  sort(all(E)); sort(y, y + m); m = unique(y, y + m, cmp_eq) - y;
  double last = E[0].first, r = 0, dy = 0;
  segtree T(0, m-1);
  for (int i = 0; i < 2*n; i++) {
    int k = E[i].second; bool in = (k >= 0); if (!in) k = ~k;
    double dx = E[i].first - last;
    r += dx * T.a;
    int a = lower_bound(y, y + m, R[k].y1, cmp_lt) - y;
    int b = lower_bound(y, y + m, R[k].y2, cmp_lt) - y;
    if (in) T.insert(a, b);
    else T.erase(a, b);
    r += fabs(dy - T.len);
    dy = T.len;
    last += dx;
  }
  return r;
}
```