Hanoi University of Industry
Student Olympiad in IT & ACM/ICPC

DST
Team Notebook

DST TEAM MEMBERS

Hung Bui – **duyhunghd6@gmail.com**
Hinh Le – **a_hinh@zing.vn**
Viet Pham – **phamviet_1990@zing.vn**

Faculty of Information Technology 2011
Group of HaUI Student Olympiad in IT & ACM/ICPC
Website: http://olympic.fit-haui.edu.vn

MỤC LỤC

CHAPTER I: NUMBER THEORETIC ALGORITHM 4
1) Number theoretic algorithm 4
2) Gauss-Jordan 6
3) Reduced Row Echelon Form 7
4) Fast Fourier transform 9
5) Simplex algorithm 10
6) Prime lower than N 12
CHAPTER II: TỐI ƯU TỔ HỢP 15
7) Dinic's Algorithm 15
8) Min cost max-flow 16
9) Push relabel Ford-Fulkerson 18
10) Min Cost Matching 20
11) Max Bipartite Matching 22
12) Lát cắt trên đồ thị 22
13) Graph Cut Inference 23
14) Ford-Fulkerson 26
CHAPTER III: GRAPH 26
1) GRAPH PRESENTATION 26
a) Using Adjency List 26
b) Danh sách kề có trọng số 27
2) GRAPH SEARCHING 29
a) Depth First Search 29
b) Breath First Search 30
3) FIND MIN PATH 32
a) Thuật toán FLOYD 32
b) Thuật toán DIJKSTRA 33
c) DIJKSTRA using Adjancy List 34
4) Strong connected Component 35
CHAPTER IV: STRING 37
1) STRING MATCHING 37
a) Knuth-Morris-Pratt 37
b) Boyer-More 39
CHAPTER V: DYNAMIC PROGRAMMING 42

1) CÁC BÀI TOÁN QUY HOẠCH ĐỘNG ĐIỂN HÌNH 42
a) Dãy con đơn điệu dài nhất 42
b) Longest Increasing Subsequence Code O(NlogN) 43
c) Vali (B) 45
d) Biến đổi xâu: 47
e) Vali (A) 49
f) Nhân ma trận 51
g) Ghép cặp 52
2) QUY HOẠCH ĐỘNG TRẠNG THÁI 53
a) Trò chơi trên lưới 53
3) MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG BỔ XUNG 54
CHAPTER VI: GEOMETRY 56
1) Convex Hull 56
2) Geometry Computation 57
3) JavaGeometry 62
4) Geometry 3D 64
5) Delaunay triangulation 65
CHAPTER VII: SPECIAL DATA STRUCTURE 67
1) Binary index tree 67
2) Disjoint-Set 67
3) KD-Tree 67
4) SuffixArray 70
5) Disjoint-Set 72
6) Interval tree 73

CHAPTER I: NUMBER THEORETIC ALGORITHM

1) Number theoretic algorithm

```
// This is a collection of useful code for solving problems that
// involve modular linear equations.  Note that all of the
// algorithms described here work on nonnegative integers.

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> VI;
typedef pair<int,int> PII;

// return a % b (positive value)
int mod(int a, int b) {
  return ((a%b)+b)%b;
}

// computes gcd(a,b)
int gcd(int a, int b) {
  int tmp;
  while(b){a%=b; tmp=a; a=b; b=tmp;}
  return a;
}

// computes lcm(a,b)
int lcm(int a, int b) {
  return a/gcd(a,b)*b;
}

// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
    int q = a/b;
    int t = b; b = a%b; a = t;
    t = xx; xx = x-q*xx; x = t;
    t = yy; yy = y-q*yy; y = t;
  }
  return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
  int x, y;
  VI solutions;
  int d = extended_euclid(a, n, x, y);
  if (!(b%d)) {
    x = mod (x*(b/d), n);
    for (int i = 0; i < d; i++)
      solutions.push_back(mod(x + i*(n/d), n));
  }
  return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
  int x, y;
  int d = extended_euclid(a, n, x, y);
  if (d > 1) return -1;
```

```
  return mod(x,n);
}

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.
PII chinese_remainder_theorem(int x, int a, int y, int b) {
  int s, t;
  int d = extended_euclid(x, y, s, t);
  if (a%d != b%d) return make_pair(0, -1);
  return make_pair(mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &x, const VI &a) {
  PII ret = make_pair(a[0], x[0]);
  for (int i = 1; i < x.size(); i++) {
    ret = chinese_remainder_theorem(ret.first, ret.second, x[i], a[i]);
    if (ret.second == -1) break;
  }
  return ret;
}

// computes x and y such that ax + by = c; on failure, x = y =-1
void linear_diophantine(int a, int b, int c, int &x, int &y) {
  int d = gcd(a,b);
  if (c%d) {
    x = y = -1;
  } else {
    x = c/d * mod_inverse(a/d, b/d);
    y = (c-a*x)/b;
  }
}

int main() {

  // expected: 2
  cout << gcd(14, 30) << endl;

  // expected: 2 -2 1
  int x, y;
  int d = extended_euclid(14, 30, x, y);
  cout << d << " " << x << " " << y << endl;

  // expected: 95 45
  VI sols = modular_linear_equation_solver(14, 30, 100);
  for (int i = 0; i < (int) sols.size(); i++) cout << sols[i] << " ";
  cout << endl;

  // expected: 8
  cout << mod_inverse(8, 9) << endl;

  // expected: 23 56
  //           11 12
  int xs[] = {3, 5, 7, 4, 6};
  int as[] = {2, 3, 2, 3, 5};
  PII ret = chinese_remainder_theorem(VI (xs, xs+3), VI(as, as+3));
  cout << ret.first << " " << ret.second << endl;
  ret = chinese_remainder_theorem (VI(xs+3, xs+5), VI(as+3, as+5));
  cout << ret.first << " " << ret.second << endl;
```

```
  // expected: 5 -15
  linear_diophantine(7, 2, 5, x, y);
  cout << x << " " << y << endl;

}
```

2) Gauss-Jordan

```
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
// OUTPUT:   X      = an nxm matrix (stored in b[][])
//           A^{-1} = an nxn matrix (stored in a[][])
//           returns determinant of a[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;

  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
      for (int k = 0; k < n; k++) if (!ipiv[k])
        if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj;
    icol[i] = pk;

    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p] *= c;
    for (int p = 0; p < m; p++) b[pk][p] *= c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
```

```
    }
  }

  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
  }

  return det;
}

int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
  double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
  VVT a(n), b(n);
  for (int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }

  double det = GaussJordan(a, b);

  // expected: 60
  cout << "Determinant: " << det << endl;

  // expected: -0.233333 0.166667 0.133333 0.0666667
  //           0.166667 0.166667 0.333333 -0.333333
  //           0.233333 0.833333 -0.133333 -0.0666667
  //           0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

  // expected: 1.63333 1.3
  //           -0.166667 0.5
  //           2.36667 1.7
  //           -1.85 -1.35
  cout << "Solution: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

3) Reduced Row Echelon Form

```
// Reduced row echelon form via Gauss-Jordan elimination
// with partial pivoting.  This can be used for computing
// the rank of a matrix.
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//
// OUTPUT:   rref[][] = an nxm matrix (stored in a[][])
//           returns rank of a[][]

#include <iostream>
#include <vector>
#include <cmath>
```

```
using namespace std;

const double EPSILON = 1e-10;

typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

int rref(VVT &a) {
  int n = a.size();
  int m = a[0].size();
  int r = 0;
  for (int c = 0; c < m; c++) {
    int j = r;
    for (int i = r+1; i < n; i++)
      if (fabs(a[i][c]) > fabs(a[j][c])) j = i;
    if (fabs(a[j][c]) < EPSILON) continue;
    swap(a[j], a[r]);

    T s = 1.0 / a[r][c];
    for (int j = 0; j < m; j++) a[r][j] *= s;
    for (int i = 0; i < n; i++) if (i != r) {
      T t = a[i][c];
      for (int j = 0; j < m; j++) a[i][j] -= t * a[r][j];
    }
    r++;
  }
  return r;
}

int main(){
  const int n = 5;
  const int m = 4;
  double A[n][m] = { {16,2,3,13},{5,11,10,8},{9,7,6,12},{4,14,15,1},{13,21,21,13}
};
  VVT a(n);
  for (int i = 0; i < n; i++)
    a[i] = VT(A[i], A[i] + n);

  int rank = rref (a);

  // expected: 4
  cout << "Rank: " << rank << endl;

  // expected: 1 0 0 1
  //           0 1 0 3
  //           0 0 1 -3
  //           0 0 0 2.78206e-15
  //           0 0 0 3.22398e-15
  cout << "rref: " << endl;
  for (int i = 0; i < 5; i++){
    for (int j = 0; j < 4; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

}
```

4) Fast Fourier transform

```
// Convolution using the fast Fourier transform (FFT).
//
// INPUT:
//     a[1...n]
//     b[1...m]
//
// OUTPUT:
//     c[1...n+m-1] such that c[k] = sum_{i=0}^k a[i] b[k-i]
//
// Alternatively, you can use the DFT() routine directly, which will
// zero-pad your input to the next largest power of 2 and compute the
// DFT or inverse DFT.

#include <iostream>
#include <vector>
#include <complex>

using namespace std;

typedef long double DOUBLE;
typedef complex<DOUBLE> COMPLEX;
typedef vector<DOUBLE> VD;
typedef vector<COMPLEX> VC;

struct FFT {
  VC A;
  int n, L;

  int ReverseBits(int k) {
    int ret = 0;
    for (int i = 0; i < L; i++) {
      ret = (ret << 1) | (k & 1);
      k >>= 1;
    }
    return ret;
  }

  void BitReverseCopy(VC a) {
    for (n = 1, L = 0; n < a.size(); n <<= 1, L++) ;
    A.resize(n);
    for (int k = 0; k < n; k++)
      A[ReverseBits(k)] = a[k];
  }

  VC DFT(VC a, bool inverse) {
    BitReverseCopy(a);
    for (int s = 1; s <= L; s++) {
      int m = 1 << s;
      COMPLEX wm = exp(COMPLEX(0, 2.0 * M_PI / m));
      if (inverse) wm = COMPLEX(1, 0) / wm;
      for (int k = 0; k < n; k += m) {
        COMPLEX w = 1;
        for (int j = 0; j < m/2; j++) {
          COMPLEX t = w * A[k + j + m/2];
          COMPLEX u = A[k + j];
          A[k + j] = u + t;
          A[k + j + m/2] = u - t;
          w = w * wm;
        }
      }
    }
    if (inverse) for (int i = 0; i < n; i++) A[i] /= n;
    return A;
```

```
  }

  // c[k] = sum_{i=0}^k a[i] b[k-i]
  VD Convolution(VD a, VD b) {
    int L = 1;
    while ((1 << L) < a.size()) L++;
    while ((1 << L) < b.size()) L++;
    int n = 1 << (L+1);

    VC aa, bb;
    for (size_t i = 0; i < n; i++) aa.push_back(i < a.size() ? COMPLEX(a[i], 0) :
0);
    for (size_t i = 0; i < n; i++) bb.push_back(i < b.size() ? COMPLEX(b[i], 0) :
0);

    VC AA = DFT(aa, false);
    VC BB = DFT(bb, false);
    VC CC;
    for (size_t i = 0; i < AA.size(); i++) CC.push_back(AA[i] * BB[i]);
    VC cc = DFT(CC, true);

    VD c;
    for (int i = 0; i < a.size() + b.size() - 1; i++) c.push_back(cc[i].real());
    return c;
  }

};

int main() {
  double a[] = {1, 3, 4, 5, 7};
  double b[] = {2, 4, 6};

  FFT fft;
  VD c = fft.Convolution(VD(a, a + 5), VD(b, b + 3));

  // expected output: 2 10 26 44 58 58 42
  for (int i = 0; i < c.size(); i++) cerr << c[i] << " ";
  cerr << endl;

  return 0;
}
```

5) Simplex algorithm

```
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <limits>
```

```
using namespace std;

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;

  LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, VD(n+2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m+1][n] = 1;
  }

  void Pivot(int r, int s) {
    for (int i = 0; i < m+2; i++) if (i != r)
      for (int j = 0; j < n+2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] / D[r][s];
    for (int j = 0; j < n+2; j++) if (j != s) D[r][j] /= D[r][s];
    for (int i = 0; i < m+2; i++) if (i != r) D[i][s] /= -D[r][s];
    D[r][s] = 1.0 / D[r][s];
    swap(B[r], N[s]);
  }

  bool Simplex(int phase) {
    int x = phase == 1 ? m+1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s]) s =
j;
      }
      if (D[x][s] >= -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] <= 0) continue;
        if (r == -1 || D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s] ||
            D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s] && B[i] < B[r]) r = i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }

  DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] <= -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m+1][n+1] < -EPS) return -
numeric_limits<DOUBLE>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
```

```
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] < N[s]) s
= j;
        Pivot(i, s);
      }
    }
    if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n+1];
    return D[m][n+1];
  }
};

int main() {

  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };

  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);

  cerr << "VALUE: "<< value << endl;
  cerr << "SOLUTION:";
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;
}
```

6) Prime lower than N

```
// O(sqrt(x)) Exhaustive Primality Test
#include <cmath>
#define EPS 1e-7
typedef long long LL;
bool IsPrimeSlow (LL x)
{
  if(x<=1) return false;
  if(x<=3) return true;
  if (!(x%2) || !(x%3)) return false;
  LL s=(LL)(sqrt((double)(x))+EPS);
  for(LL i=5;i<=s;i+=6)
  {
    if (!(x%i) || !(x%(i+2))) return false;
  }
  return true;
}
// Primes less than 1000:
//       2     3     5     7    11    13    17    19    23    29    31    37
//      41    43    47    53    59    61    67    71    73    79    83    89
//      97   101   103   107   109   113   127   131   137   139   149   151
//     157   163   167   173   179   181   191   193   197   199   211   223
//     227   229   233   239   241   251   257   263   269   271   277   281
//     283   293   307   311   313   317   331   337   347   349   353   359
```

```
//    367   373   379   383   389   397   401   409   419   421   431   433
//    439   443   449   457   461   463   467   479   487   491   499   503
//    509   521   523   541   547   557   563   569   571   577   587   593
//    599   601   607   613   617   619   631   641   643   647   653   659
//    661   673   677   683   691   701   709   719   727   733   739   743
//    751   757   761   769   773   787   797   809   811   821   823   827
//    829   839   853   857   859   863   877   881   883   887   907   911
//    919   929   937   941   947   953   967   971   977   983   991   997

// Other primes:
//    The largest prime smaller than 10 is 7.
//    The largest prime smaller than 100 is 97.
//    The largest prime smaller than 1000 is 997.
//    The largest prime smaller than 10000 is 9973.
//    The largest prime smaller than 100000 is 99991.
//    The largest prime smaller than 1000000 is 999983.
//    The largest prime smaller than 10000000 is 9999991.
//    The largest prime smaller than 100000000 is 99999989.
//    The largest prime smaller than 1000000000 is 999999937.
//    The largest prime smaller than 10000000000 is 9999999967.
//    The largest prime smaller than 100000000000 is 99999999977.
//    The largest prime smaller than 1000000000000 is 999999999989.
//    The largest prime smaller than 10000000000000 is 9999999999971.
//    The largest prime smaller than 100000000000000 is 99999999999973.
//    The largest prime smaller than 1000000000000000 is 999999999999989.
//    The largest prime smaller than 10000000000000000 is 9999999999999937.
//    The largest prime smaller than 100000000000000000 is 99999999999999997.
//    The largest prime smaller than 1000000000000000000 is 999999999999999989.
```

7) The nth Fibonacci

```
int64 fibonacci(int n)
{
    int64 fib[2][2]= {{1,1},{1,0}},ret[2][2]= {{1,0},{0,1}},tmp[2][2]=
{{0,0},{0,0}};
    int i,j,k;
    while(n)
    {
        if(n&1)
        {
            memset(tmp,0,sizeof tmp);
            for(i=0; i<2; i++) for(j=0; j<2; j++) for(k=0; k<2; k++)
                        tmp[i][j]=(tmp[i][j]+ret[i][k]*fib[k][j]);
            for(i=0; i<2; i++) for(j=0; j<2; j++) ret[i][j]=tmp[i][j];
        }
        memset(tmp,0,sizeof tmp);
        for(i=0; i<2; i++) for(j=0; j<2; j++) for(k=0; k<2; k++)
                    tmp[i][j]=(tmp[i][j]+fib[i][k]*fib[k][j]);
        for(i=0; i<2; i++) for(j=0; j<2; j++) fib[i][j]=tmp[i][j];
        n/=2;
    }
    return (ret[0][1]);
}
```

8) Read double with 100 character numbers

```
#include<iostream>
#include<cmath>
using namespace std;

int main()
{
    double n,p;
    cout.setf(ios::fixed,ios::floatfield);
```

```
    cout.precision(0);
    while(cin>>n>>p)
        cout<<pow(p,1/n)<<endl;
}
```

9) Dinic's Algorithm

```
// Adjacency list implementation of Dinic's blocking flow algorithm.
// This is very fast in practice, and only loses to push-relabel flow.
//
// Running time:
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).

#include <cmath>
#include <vector>
#include <iostream>
#include <queue>

using namespace std;

const int INF = 2000000000;

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct Dinic {
  int N;
  vector<vector<Edge> > G;
  vector<Edge *> dad;
  vector<int> Q;

  Dinic(int N) : N(N), G(N), dad(N), Q(N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

  long long BlockingFlow(int s, int t) {
    fill(dad.begin(), dad.end(), (Edge *) NULL);
    dad[s] = &G[0][0] - 1;

    int head = 0, tail = 0;
    Q[tail++] = s;
    while (head < tail) {
      int x = Q[head++];
      for (int i = 0; i < G[x].size(); i++) {
        Edge &e = G[x][i];
        if (!dad[e.to] && e.cap - e.flow > 0) {
          dad[e.to] = &G[x][i];
          Q[tail++] = e.to;
        }
      }
    }
```

```
    if (!dad[t]) return 0;

    long long totflow = 0;
    for (int i = 0; i < G[t].size(); i++) {
      Edge *start = &G[G[t][i].to][G[t][i].index];
      int amt = INF;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
        if (!e) { amt = 0; break; }
        amt = min(amt, e->cap - e->flow);
      }
      if (amt == 0) continue;
      for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
        e->flow += amt;
        G[e->to][e->index].flow -= amt;
      }
      totflow += amt;
    }
    return totflow;
  }

  long long GetMaxFlow(int s, int t) {
    long long totflow = 0;
    while (long long flow = BlockingFlow(s, t))
      totflow += flow;
    return totflow;
  }
};
```

10) Min cost max-flow

```
// Implementation of min cost max flow algorithm using adjacency
// matrix (Edmonds and Karp 1972).  This implementation keeps track of
// forward and reverse edges separately (so you can set cap[i][j] !=
// cap[j][i]).  For a regular max flow, set all edge costs to 0.
//
// Running time, O(|V|^2) cost per augmentation
//     max flow:           O(|V|^3) augmentations
//     min cost max flow:  O(|V|^4 * MAX_EDGE_COST) augmentations
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - (maximum flow value, minimum cost value)
//     - To obtain the actual flow, look at positive values only.

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
  int N;
```

```
  VVL cap, flow, cost;
  VI found;
  VL dist, pi, width;
  VPII dad;

  MinCostMaxFlow(int N) :
    N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
    found(N), dist(N), pi(N), width(N), dad(N) {}

  void AddEdge(int from, int to, L cap, L cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
  }

  void Relax(int s, int k, L cap, L cost, int dir) {
    L val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }

  L Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
      int best = -1;
      found[s] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best = k;
      }
      s = best;
    }

    for (int k = 0; k < N; k++)
      pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }

  pair<L, L> GetMaxFlow(int s, int t) {
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        } else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
};
```

11) Push relabel Ford-Fulkerson

```
// Adjacency list implementation of FIFO push relabel maximum flow
// with the gap relabeling heuristic.  This implementation is
// significantly faster than straight Ford-Fulkerson.  It solves
// random problems with 10000 vertices and 1000000 edges in a few
// seconds, though it is possible to construct test cases that
// achieve the worst-case.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).

#include <cmath>
#include <vector>
#include <iostream>
#include <queue>

using namespace std;

typedef long long LL;

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct PushRelabel {
  int N;
  vector<vector<Edge> > G;
  vector<LL> excess;
  vector<int> dist, active, count;
  queue<int> Q;

  PushRelabel(int N) : N(N), G(N), excess(N), dist(N), active(N), count(2*N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

  void Enqueue(int v) {
    if (!active[v] && excess[v] > 0) { active[v] = true; Q.push(v); }
  }

  void Push(Edge &e) {
    int amt = int(min(excess[e.from], LL(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);
  }
```

```
  void Gap(int k) {
    for (int v = 0; v < N; v++) {
      if (dist[v] < k) continue;
      count[dist[v]]--;
      dist[v] = max(dist[v], N+1);
      count[dist[v]]++;
      Enqueue(v);
    }
  }

  void Relabel(int v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (int i = 0; i < G[v].size(); i++)
      if (G[v][i].cap - G[v][i].flow > 0)
        dist[v] = min(dist[v], dist[G[v][i].to] + 1);
    count[dist[v]]++;
    Enqueue(v);
  }

  void Discharge(int v) {
    for (int i = 0; excess[v] > 0 && i < G[v].size(); i++) Push(G[v][i]);
    if (excess[v] > 0) {
      if (count[dist[v]] == 1)
        Gap(dist[v]);
      else
        Relabel(v);
    }
  }

  LL GetMaxFlow(int s, int t) {
    count[0] = N-1;
    count[N] = 1;
    dist[s] = N;
    active[s] = active[t] = true;
    for (int i = 0; i < G[s].size(); i++) {
      excess[s] += G[s][i].cap;
      Push(G[s][i]);
    }

    while (!Q.empty()) {
      int v = Q.front();
      Q.pop();
      active[v] = false;
      Discharge(v);
    }

    LL totflow = 0;
    for (int i = 0; i < G[s].size(); i++) totflow += G[s][i].flow;
    return totflow;
  }
};
```

12) Min Cost Matching

```
////////////////////////////////////////////////////////////////////////////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs.  In practice, it solves 1000x1000 problems in around 1
// second.
//
//   cost[i][j] = cost for pairing left node i with right node j
//   Lmate[i] = index of right node that left node i pairs with
//   Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative.  To perform
// maximization, simply negate the cost[][] matrix.
////////////////////////////////////////////////////////////////////////////

#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());

  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
  }

  // construct primal solution satisfying complementary slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
        Lmate[i] = j;
        Rmate[j] = i;
        mated++;
        break;
      }
    }
  }

  VD dist(n);
  VI dad(n);
  VI seen(n);

  // repeat until primal solution is feasible
```

```
while (mated < n) {

  // find an unmatched left node
  int s = 0;
  while (Lmate[s] != -1) s++;

  // initialize Dijkstra
  fill(dad.begin(), dad.end(), -1);
  fill(seen.begin(), seen.end(), 0);
  for (int k = 0; k < n; k++)
    dist[k] = cost[s][k] - u[s] - v[k];

  int j = 0;
  while (true) {

    // find closest
    j = -1;
    for (int k = 0; k < n; k++) {
      if (seen[k]) continue;
      if (j == -1 || dist[k] < dist[j]) j = k;
    }
    seen[j] = 1;

    // termination condition
    if (Rmate[j] == -1) break;

    // relax neighbors
    const int i = Rmate[j];
    for (int k = 0; k < n; k++) {
      if (seen[k]) continue;
      const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
      if (dist[k] > new_dist) {
        dist[k] = new_dist;
        dad[k] = j;
      }
    }
  }

  // update dual variables
  for (int k = 0; k < n; k++) {
    if (k == j || !seen[k]) continue;
    const int i = Rmate[k];
    v[k] += dist[k] - dist[j];
    u[i] -= dist[k] - dist[j];
  }
  u[s] += dist[j];

  // augment along path
  while (dad[j] >= 0) {
    const int d = dad[j];
    Rmate[j] = Rmate[d];
    Lmate[Rmate[j]] = j;
    j = d;
  }
  Rmate[j] = s;
  Lmate[s] = j;

  mated++;
}

double value = 0;
for (int i = 0; i < n; i++)
  value += cost[i][Lmate[i]];

return value;
```

```
}
```

13) Max Bipartite Matching

```
// This code performs maximum bipartite matching.
//
// Running time: O(|E| |V|) -- often much faster in practice
//
//   INPUT: w[i][j] = edge between row node i and column node j
//   OUTPUT: mr[i] = assignment for row node i, -1 if unassigned
//           mc[j] = assignment for column node j, -1 if unassigned
//           function returns number of matches made

#include <vector>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen) {
  for (int j = 0; j < w[i].size(); j++) {
    if (w[i][j] && !seen[j]) {
      seen[j] = true;
      if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
        mr[i] = j;
        mc[j] = i;
        return true;
      }
    }
  }
  return false;
}

int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
  mr = VI(w.size(), -1);
  mc = VI(w[0].size(), -1);

  int ct = 0;
  for (int i = 0; i < w.size(); i++) {
    VI seen(w[0].size());
    if (FindMatch(i, w, mr, mc, seen)) ct++;
  }
  return ct;
}
```

14) Lát cắt trên đồ thị

```
// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//
// OUTPUT:
//     - (min cut value, nodes in half of min cut)

#include <cmath>
#include <vector>
#include <iostream>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
```

```
const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
  int N = weights.size();
  VI used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    VI w = weights[0];
    VI added = used;
    int prev, last = 0;
    for (int i = 0; i < phase; i++) {
      prev = last;
      last = -1;
      for (int j = 1; j < N; j++)
        if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
      if (i == phase-1) {
        for (int j = 0; j < N; j++) weights[prev][j] += weights[last][j];
        for (int j = 0; j < N; j++) weights[j][prev] = weights[prev][j];
        used[last] = true;
        cut.push_back(last);
        if (best_weight == -1 || w[last] < best_weight) {
          best_cut = cut;
          best_weight = w[last];
        }
      } else {
        for (int j = 0; j < N; j++)
          w[j] += weights[last][j];
        added[last] = true;
      }
    }
  }
  return make_pair(best_weight, best_cut);
}
```

15) Graph Cut Inference

```
// Special-purpose {0,1} combinatorial optimization solver for
// problems of the following by a reduction to graph cuts:
//
//        minimize         sum_i  psi_i(x[i])
//  x[1]...x[n] in {0,1}      + sum_{i < j}  phi_{ij}(x[i], x[j])
//
// where
//      psi_i : {0, 1} --> R
//   phi_{ij} : {0, 1} x {0, 1} --> R
//
// such that
//   phi_{ij}(0,0) + phi_{ij}(1,1) <= phi_{ij}(0,1) + phi_{ij}(1,0)   (*)
//
// This can also be used to solve maximization problems where the
// direction of the inequality in (*) is reversed.
//
// INPUT: phi -- a matrix such that phi[i][j][u][v] = phi_{ij}(u, v)
//        psi -- a matrix such that psi[i][u] = psi_i(u)
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution
//
// To use this code, create a GraphCutInference object, and call the
// DoInference() method.  To perform maximization instead of minimization,
// ensure that #define MAXIMIZATION is enabled.

#include <vector>
#include <iostream>
```

```
using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef vector<VVI> VVVI;
typedef vector<VVVI> VVVVI;

const int INF = 1000000000;

// comment out following line for minimization
#define MAXIMIZATION

struct GraphCutInference {
  int N;
  VVI cap, flow;
  VI reached;

  int Augment(int s, int t, int a) {
    reached[s] = 1;
    if (s == t) return a;
    for (int k = 0; k < N; k++) {
      if (reached[k]) continue;
      if (int aa = min(a, cap[s][k] - flow[s][k])) {
        if (int b = Augment(k, t, aa)) {
          flow[s][k] += b;
          flow[k][s] -= b;
          return b;
        }
      }
    }
    return 0;
  }

  int GetMaxFlow(int s, int t) {
    N = cap.size();
    flow = VVI(N, VI(N));
    reached = VI(N);

    int totflow = 0;
    while (int amt = Augment(s, t, INF)) {
      totflow += amt;
      fill(reached.begin(), reached.end(), 0);
    }
    return totflow;
  }

  int DoInference(const VVVVI &phi, const VVI &psi, VI &x) {
    int M = phi.size();
    cap = VVI(M+2, VI(M+2));
    VI b(M);
    int c = 0;

    for (int i = 0; i < M; i++) {
      b[i] += psi[i][1] - psi[i][0];
      c += psi[i][0];
      for (int j = 0; j < i; j++)
        b[i] += phi[i][j][1][1] - phi[i][j][0][1];
      for (int j = i+1; j < M; j++) {
        cap[i][j] = phi[i][j][0][1] + phi[i][j][1][0] - phi[i][j][0][0] -
phi[i][j][1][1];
        b[i] += phi[i][j][1][0] - phi[i][j][0][0];
        c += phi[i][j][0][0];
      }
    }
```

```
#ifdef MAXIMIZATION
    for (int i = 0; i < M; i++) {
      for (int j = i+1; j < M; j++)
        cap[i][j] *= -1;
      b[i] *= -1;
    }
    c *= -1;
#endif

    for (int i = 0; i < M; i++) {
      if (b[i] >= 0) {
        cap[M][i] = b[i];
      } else {
        cap[i][M+1] = -b[i];
        c += b[i];
      }
    }

    int score = GetMaxFlow(M, M+1);
    fill(reached.begin(), reached.end(), 0);
    Augment(M, M+1, INF);
    x = VI(M);
    for (int i = 0; i < M; i++) x[i] = reached[i] ? 0 : 1;
    score += c;
#ifdef MAXIMIZATION
    score *= -1;
#endif

    return score;
  }

};

int main() {

  // solver for "Cat vs. Dog" from NWERC 2008

  int numcases;
  cin >> numcases;
  for (int caseno = 0; caseno < numcases; caseno++) {
    int c, d, v;
    cin >> c >> d >> v;

    VVVVI phi(c+d, VVVI(c+d, VVI(2, VI(2))));
    VVI psi(c+d, VI(2));
    for (int i = 0; i < v; i++) {
      char p, q;
      int u, v;
      cin >> p >> u >> q >> v;
      u--; v--;
      if (p == 'C') {
        phi[u][c+v][0][0]++;
        phi[c+v][u][0][0]++;
      } else {
        phi[v][c+u][1][1]++;
        phi[c+u][v][1][1]++;
      }
    }

    GraphCutInference graph;
    VI x;
    cout << graph.DoInference(phi, psi, x) << endl;
  }
```

```
  return 0;
}
```

16) Ford-Fulkerson

```
//Maximum flow using Ford-Fulkerson
#include<memory.h>


#define min(a, b) (((a)<(b))?(a):(b))
#define INF 1000000000

//global variables
int n;
int cap[MAXN][MAXN];
bool v[MAXN];
int source, sink;

//returns the capacity of the path. zero if there isn't one
int augment(int x, int minedge)
{
  if(x==sink) return minedge;
  v[x]=true;
  for(int i=0;i<n;i++)
  {
    if(!v[i] && cap[x][i])
    {
      int ret=augment(i, min(minedge, cap[x][i]));
      if(ret){cap[i][x]-=ret; cap[x][i]+=ret; return ret;}
    }
  }
  return 0;
}

//returns the maximum flow
int maxFlow()
{
  int ret=0;
  while(true)
  {
    memset(v, false, sizeof(v));
    int flow=augment(source, INF);
    if(!flow) break;
    ret+=flow;
  }
  return ret;
}
```

CHAPTER III: GRAPH

1) GRAPH PRESENTATION

a) Using Adjency List

```
#include <conio.h>
#include <iostream>
#include <list>
using namespace std;
main()
{
  freopen("G.inp","r",stdin);
  list<int> G[1000];
  int n;
  cin>>n;
  int tg;
  int v;
  for(int i=1;i<=n;i++)
  {
```

```
    cin>>tg;
    for(int j=0;j<tg;j++)
    {
      cin>>v;
      G[i].push_back(v);
    }
  }

  for(int i = 1; i<=n; i++)
  {
    for(list<int>::iterator vii=G[i].begin();vii!=G[i].end();vii++)
      cout<<*vii<<" ";
    cout<<endl;
  }
  getch();
}
```

Samples Input

```
5
2 2 3
3 1 4 5
3 1 4 5
2 2 3
2 2 3
```

b) Danh sách kề có trọng số

```
#include <iostream>
#include <list>
#define MAXN 1001

using namespace std;

struct Node
{
    int x;
    int dinh;
};
list<Node> Graph[MAXN];

int G[MAXN][MAXN];
int N;

main()
{
    freopen("BFS1.INP", "r", stdin);
    freopen("BFS_adj.OUT", "w", stdout);
    cin>>N;
    for(int i = 1; i<=N-1; i++)
        for(int j = i+1; j<=N; j++)
        {
            int x;
            cin>>x;
            if(x>=0)
            {
                Node d;
                d.x = x;
                d.dinh = j;
                Graph[i].push_front(d);
            }
        }

    for(int i = 1; i<=N; i++)
    {
        cout<<"Dinh thu "<<i<<": ";
        for(list<Node>::iterator itor = Graph[i].begin(); itor!=Graph[i].end(); it
or++)
```

```
            {
                cout<<"("<<i<<", "<<(*itor).dinh<<") = "<<(*itor).x<<" -> ";
            }
            cout<<endl;
        }
    }
```

2) GRAPH SEARCHING

a) Depth First Search

```
#include <iostream>
#include <vector>
#include <fstream>
#define NMAX 1000

using namespace std;

int N, s, e;
int A[NMAX][NMAX];
bool CX[NMAX];
vector<int> vet;

void print()
{
   cout<<"Vet tim duoc la: ";
   for(vector<int>::iterator itor=vet.begin(); itor!=vet.end(); itor++)
      cout<<*itor<<" ";
   cout<<endl;
}

void DFS(int start, int end)
{
   //Neu den dich thi in ra danh sach duong di
   if(start==end)
   {
      print();
   }
   else
   {
      //Duyet qua tat ca cac dinh
      for(int j=1; j<=N; j++)
      {
         //Neu dinh j ke voi start va chua duoc xet. ta xet dinh j va day vao VECTOR, ket thuc DFS tai dinh di, ta se quay lui
         if(A[start][j]==1 && !CX[j])
         {
            CX[j]=true;
            vet.push_back(j);
            DFS(j, end);
            vet.pop_back();
            CX[j]=false;
         }
      }
   }
}

main()
{
   freopen("DFS1.INP", "r", stdin);
   cin>>N>>s>>e;
   for(int i=1 ; i <= N-1 ; i++)
      for(int j = i+1 ; j <= N; j++)
      {
         cin>>A[i][j];
         A[j][i]=A[i][j];
      }
   //In ra ma tran ke
   for(int i=1 ; i <= N-1 ; i++)
   {
      for(int j = i+1 ; j <= N; j++)
      {
         cout<<A[i][j]<<" ";
      }
      cout<<endl;
   }

   //Them phan tu bat dau vao VECTOR
   CX[s]=true;
```

```
    vet.push_back(s);
    //DFS dinh dau tien
    DFS(s, e);
}
```

Samples Input

```
7 1 7
1 1 0 0 0 0
1 0 0 1 0
1 1 0 0
1 0 1
1 0
1
```

b) Breath First Search

```
#include <iostream>
#include <vector>
#include <fstream>
#include <queue>
#define NMAX 1000

using namespace std;

int N, s, e;
int A[NMAX][NMAX];
bool CX[NMAX];
int Trace[NMAX], Length[NMAX];

queue<int> que;

void print()
{
    cout<<"Duong di tim duoc la: ";
    int k = e;
    do
    {
        cout<<k<<" ";
        k = Trace[k];
    }while(k!=s);
    cout<<s<<" ";
}

void BFS(int end)
{
    while(!que.empty())
    {
        int start = que.front();
        que.pop();

        //Neu den dich thi in ra danh sach duong di
        if(start==end)
        {
            print();
            break;
        }
        else
        {
            //Duyet qua tat ca cac dinh
            for(int j=1; j<=N; j++)
                if(A[start][j]==1 && !CX[j])
                {
                    que.push(j);
                    Trace[j] = start;
                    Length[j] = Length[start+1];
                    CX[j] = true;
```

```
                }
            }
        }
}

main()
{
    freopen("BFS1.INP", "r", stdin);
    freopen("BFS1.OUT", "w", stdout);
    cin>>N>>s>>e;
    for(int i=1 ; i <= N-1 ; i++)
        for(int j = i+1 ; j <= N; j++)
        {
            cin>>A[i][j];
            A[j][i]=A[i][j];
        }
    //In ra ma tran ke
    for(int i=1 ; i <= N ; i++)
    {
        for(int j = 1 ; j <= N; j++)
        {
            cout<<A[i][j]<<" ";
        }
        cout<<endl;
    }

    //DFS dinh dau tien
    CX[s] = true;
    que.push(s);
    BFS(e);
}
```

Samples Input

```
7 1 7
1 1 0 0 0 0
1 0 0 1 0
1 1 0 0
1 0 1
1 0
1
```

3) FIND MIN PATH

a) Thuật toán FLOYD

```
#include <iostream>
#include <fstream>
#include <vector>

#define MAXN 1001

using namespace std;

int A[MAXN][MAXN];
int Trace[MAXN][MAXN];
int N, M, K;
vector<int> vet;

void print()
{
   for(int i = vet.size()-1; i>=0; i--)
      cout<<vet[i]<<" ";
   cout<<endl;
   vet.erase(vet.begin(), vet.end());
}


void truyvet(int u, int v)
{
   do
   {
      vet.push_back(v);
      v=Trace[u][v];
   }while(u!=v);
   vet.push_back(v);
   print();
}



main()
{
   freopen("FLOYD1.INP", "r", stdin);
   freopen("FLOYD1.OUT", "w", stdout);

   //Doc do thi tu file
//    cin>>N>>M>>K;
   cin>>N;
   for(int i = 1; i<N; i++)
      for(int j = i+1; j<=N; j++)
      {
         int x = 0;
         cin>>x;
         if(x==-1) x=9999999;
         A[i][j] = A[j][i] = x;
         if(9999999!=x)
         {
            Trace[i][j]=i;
            Trace[j][i] = j;
         }
      }


   //Thuat toan Floyd
   for(int i = 1; i<=N; i++)
   {
      for(int j = 1; j<=N; j++)
      {
         for(int k = 1; k<=N; k++)
         {
            if(A[i][k]+A[k][j]<A[i][j])
            {
```

```
                A[j][i] = A[i][j] = A[i][k]+A[k][j];
                Trace[i][j]=Trace[j][i]=k;
            }
        }
    }
}

cout<<"Duong di ngan nhat tu 1->7 = "<<A[1][7];

//In ra duong di ngan nhat tu dinh 1 den dinh 7
truyvet(1, 7);

}

}
```

Input samples:

```
7
4 5 -1 -1 -1 -1
-1 -1 6 -1 -1
3 1 -1 -1
-1 2 7
8 -1
7
```

b) Thuật toán DIJKSTRA

```
#include <iostream>
#include <vector>

#define MAXN 1001
#define MAXX 999999999

using namespace std;

int A[MAXN][MAXN];
int MinPath[MAXN];
int From[MAXN];
bool Free[MAXN];

int M, N, S, E;


void init()
{
    freopen("DIJKSTRA.INP", "r", stdin);
    freopen("DIJKSTRA.OUT", "w", stdout);

    //Doc file input
    scanf("%d %d %d %d", &N, &M, &S, &E);

    for(int i = 1; i<=M; i++)
    {
        int u, v, p;
        scanf("%d %d %d", &u, &v, &p);
        A[u][v] = A[v][u] = p;
    }

    //Gan duong di ngan nhat = MAXX
    for(int i = 1; i<=N; i++) MinPath[i] = MAXX;
    MinPath[S] = 0;
}


void DIJKSTRA()
{
    int g = S, minD;
    do
    {
        g=E;
        for(int i = 1; i<=N; i++)
```

```
            if(Free[i] == false && MinPath[g] > MinPath[i])
            {
               g = i;
            }
         Free[g] = true;     //Gan nhan cho dinh G co' dinh

         if(MinPath[g] == MAXX ||  g==E) break;


         for(int v = 1; v<=N; v++)
         {
            if(A[g][v] > 0 && !Free[v])
            {
               if(A[g][v] + MinPath[g] < MinPath[v])
               {
                  MinPath[v] = A[g][v] + MinPath[g];
                  From[v] = g;
               }
            }
         }

      }
      while(true);
}


void TruyVet(int end)
{
   int u = end;
   vector<int> vet;
   while(u!=S)
   {
      vet.push_back(u);
      u = From[u];
   }
   vet.push  back(S);
   printf("\nVet tim duoc: ");
   for(int i = vet.size()-1; i>=0; i--) printf("%3d", vet[i]);
   printf("\n");
}


void print()
{
   if(MinPath[E] == MAXX)
      printf("Khong co duong di!");
   else
      printf("Duong di ngan nhat la: %d", MinPath[E]);
}

main()
{
   init();
   DIJKSTRA();
   print();
   TruyVet(E);
}
```

c) DIJKSTRA using Adjancy List

```
// Implementation of Dijkstra's algorithm using adjacency lists
// and priority queue for efficiency.
//
// Running time: O(|E| log |V|)

#include <queue>
#include <stdio.h>

using namespace std;
const int INF = 2000000000;
typedef pair<int,int> PII;
```

```
int main(){

  int N, s, t;
  scanf ("%d%d%d", &N, &s, &t);
  vector<vector<PII> > edges(N);
  for (int i = 0; i < N; i++){
    int M;
    scanf ("%d", &M);
    for (int j = 0; j < M; j++){
      int vertex, dist;
      scanf ("%d%d", &vertex, &dist);
      edges[i].push_back (make_pair (dist, vertex)); // note order of arguments
here
    }
  }

  // use priority queue in which top element has the "smallest" priority
  priority_queue<PII, vector<PII>, greater<PII> > Q;
  vector<int> dist(N, INF), dad(N, -1);
  Q.push (make_pair (0, s));
  dist[s] = 0;
  while (!Q.empty()){
    PII p = Q.top();
    if (p.second == t) break;
    Q.pop();

    int here = p.second;
    for (vector<PII>::iterator it=edges[here].begin(); it!=edges[here].end();
it++){
      if (dist[here] + it->first < dist[it->second]){
        dist[it->second] = dist[here] + it->first;
        dad[it->second] = here;
        Q.push (make_pair (dist[it->second], it->second));
      }
    }
  }

  printf ("%d\n", dist[t]);
  if (dist[t] < INF)
    for(int i=t;i!=-1;i=dad[i])
      printf ("%d%c", i, (i==s?'\n':' '));

  return 0;
}
```

4) Strong connected Component

```
#include<memory.h>
struct edge{int e, nxt;};
int V, E;
edge e[MAXE], er[MAXE];
int sp[MAXV], spr[MAXV];
int group_cnt, group_num[MAXV];
bool v[MAXV];
int stk[MAXV];
void fill_forward(int x)
{
        int i;
        v[x]=true;
        for(i=sp[x];i;i=e[i].nxt) if(!v[e[i].e]) fill_forward(e[i].e);
        stk[++stk[0]]=x;
}
void fill_backward(int x)
{
        int i;
```

```
        v[x]=false;
        group_num[x]=group_cnt;
        for(i=spr[x];i;i=er[i].nxt) if(v[er[i].e]) fill_backward(er[i].e);
}
void add_edge(int v1, int v2) //add edge v1->v2
{
        e [++E].e=v2; e [E].nxt=sp [v1]; sp [v1]=E;
        er[  E].e=v1; er[E].nxt=spr[v2]; spr[v2]=E;
}
void SCC()
{
        int i;
        stk[0]=0;
        memset(v, false, sizeof(v));
        for(i=1;i<=V;i++) if(!v[i]) fill_forward(i);
        group_cnt=0;
        for(i=stk[0];i>=1;i--) if(v[stk[i]]){group_cnt++; fill_backward(stk[i]);}
}
```

CHAPTER IV: STRING

1) STRING MATCHING

a) Knuth-Morris-Pratt

```
using System;
namespace Knuth_Morris_Pratt
{
    class Test
    {
        private char[] s;
        private char[] p;

        public void Nhap()
        {
            Console.Write("Nhap xau S:");
            string stdin1 = Console.ReadLine();
            s = new char[stdin1.Length];
            s = stdin1.ToCharArray();
            Console.Write("Nhap xau P:");
            string stdin2 = Console.ReadLine();
            p = new char[stdin2.Length];
            p = stdin2.ToCharArray();
        }
        private int[] compute_MP_map()   // map[i]=| border(p[0..i-1]) |
        {
            int m = p.Length;   // p is the input pattern
            int[] MP_map = new int[m + 1];     // lưu ý rằng ta phải tính map[0..m]
            // init
            MP_map[0] = -1;     // cái này chắc ai cũng hiểu
            // bay gio di tinh map[1..m]
            int i = 0; int j = MP_map[i];
            while (i < m)
            {
                while (j >= 0 && (p[i] != p[j])) j = MP_map[j];
                j++; i++;
                MP_map[i] = j;
            }
            return MP_map;
        }
        private int[] compute_KMP_map() //thuật toán KMP cải tiến
        {
            int m = p.Length;
            int[] KMP_map = new int[m + 1];
            int[] MP_map = compute_MP_map();
            KMP_map[0] = -1; KMP_map[m] = MP_map[m];
            for (int i = 1; i < m; i++)
            {
                int j = MP_map[i];
                if (p[i] != p[j]) KMP_map[i] = j;
                else KMP_map[i] = MP_map[j];
            }
            return KMP_map;
        }

        public void Morris_Pratt()
        {
            int[] map = compute_KMP_map();
            int n = s.Length;
            int m = p.Length;
            int i = 0;
            string res = "";
            for (int j = 0; j < n; j++)
            {
                while ((i >= 0) && (p[i] != s[j])) i = map[i];
                i++;            // Có 2 khả năng xảy ra: hoặc là đã đi hết chuỗi p (i=m-1)
hoặc là i=-1 , lợi dụng cả 2 điều này
                if (i == m)
                {
                    res += (j - m + 1).ToString() + " ";
                    i = map[i];
```

```
                }
            }
            Console.Write(res);
        }
        static void Main(string[] args)
        {
            Test object1 = new Test();
            object1.Nhap();
            object1.Morris_Pratt();
            Console.ReadLine();
        }
    }
}
```

C++ IMPLEMENT

```
#include <iostream>
#include <string>

using namespace std;

int m, n, i, j;
int Next[100000];
string a;
string b;

void PREKMP()
{
    i=0;
    j=-1;
    Next[0]=0;
    while (i<m)
    {
        while(j>-1 && b[i]!=b[j])
            j=Next[j];
        i++;    j++;
        if(b[i]==b[j])
            Next[i]=Next[j];
        else
            Next[i]=j;
    }
}
void KMP()
{
    i=0;
    j=0;
    while(j<n)
    {
        while (i>=0 && a[j]!=b[i])
            i=Next[i];
        i++;
        j++;
        if(i>m-1)
        {
            cout<<j-i+1<<" ";
            i=Next[i];
        }
    }
}
main()
{
    getline(cin,a);
    getline(cin,b);
    n=a.size();
    m=b.size();
    PREKMP();
    KMP();
}
```

KMP phiên bản 2

```
//linear time pattern matching algorithm
#include<cstdlib>
#include<cstring>

int *computePrefixFunction(char *P, int m)
{
  int *pi=(int *)malloc(m*sizeof(int));
  pi[0]=-1;
  int k=-1;
  for(int q=1;q<m;q++)
  {
    while(k>=0 && P[k+1]!=P[q]) k=pi[k];
    if(P[k+1]==P[q]) k++;
    pi[q]=k;
  }
  return pi;
}

void KMP(char *T, char *P)
{
  int n=strlen(T), m=strlen(P);
  int *pi=computePrefixFunction(P, m);
  int q=-1;
  for(int i=0;i<n;i++)
  {
    while(q>=0 && P[q+1]!=T[i]) q=pi[q];
    if(P[q+1]==T[i]) q++;
    if(q==m)
    {
      //P occurs T[i-m+1..i]. Do whatever you want here
      q=pi[q];
    }
  }
}
```

b) Boyer-More

```
#include <string>
#include <iostream>

using namespace std;

int m_badCharacterShift[256] ;
int m_goodSuffixShift[1000001] ;
int m_suffixes[1000001] ;
string m_pattern;


void BuildBadCharacterShift(string pattern)
{
  for (int c = 0; c < 256; ++c)
    m_badCharacterShift[c] = pattern.size();
  for (int i = 0; i < pattern.size() - 1; ++i)
    m_badCharacterShift[pattern[i]] = pattern.size() - i - 1;
}

void FindSuffixes(string pattern)
{
  int f = 0, g;

  int patternLength = pattern.size();

  m_suffixes[patternLength - 1] = patternLength;
  g = patternLength - 1;
  for (int i = patternLength - 2; i >= 0; --i)
```

```
  {
    if (i > g && m_suffixes[i + patternLength - 1 - f] < i - g)
      m_suffixes[i] = m_suffixes[i + patternLength - 1 - f];
    else
    {
      if (i < g)
        g = i;
      f = i;
      while (g >= 0 && (pattern[g] == pattern[g + patternLength - 1 - f]))
        --g;
      m_suffixes[i] = f - g;
    }
  }

}

void BuildGoodSuffixShift(string pattern, int suff[])
{
  int patternLength = pattern.size();

  for (int i = 0; i < patternLength; ++i)
    m_goodSuffixShift[i] = patternLength;
  int j = 0;
  for (int i = patternLength - 1; i >= -1; --i)
    if (i == -1 || suff[i] == i + 1)
      for (; j < patternLength - 1 - i; ++j)
        if (m_goodSuffixShift[j] == patternLength)
          m_goodSuffixShift[j] = patternLength - 1 - i;
  for (int i = 0; i <= patternLength - 2; ++i)
    m_goodSuffixShift[patternLength - 1 - suff[i]] = patternLength - 1 - i;

}

void setString(string pattern)
{
  m_pattern = pattern;
  BuildBadCharacterShift(pattern);
  FindSuffixes(pattern);
  BuildGoodSuffixShift(pattern, m_suffixes);
}

int min2(int a, int b)
{
    return a<b?a:b;
}
int max2(int a, int b)
{
    return a>b?a:b;
}

void TurboBoyerMooreMatch(string text, int startingIndex)
{
    int patternLength = m_pattern.size();
    int textLength = text.size();

    /* Searching */
    int index = startingIndex;
    int overlap = 0;
    int shift = patternLength;
    while (index <= textLength - patternLength)
    {
    int unmatched = patternLength - 1;

    while (unmatched >= 0 && (m_pattern[unmatched] == text[unmatched + index]))
    {
        --unmatched;
        if (overlap != 0 && unmatched == patternLength - 1 - shift)
            unmatched -= overlap;
```

```
        }

        if (unmatched < 0)
        {
            //return index;
            cout<<index+1<<" ";
            shift = m_goodSuffixShift[0];
            overlap = patternLength - shift;
        }
        else
        {
            int matched = patternLength - 1 - unmatched;
            int turboShift = overlap - matched;
            int bcShift = m_badCharacterShift[text[unmatched + index]] -
patternLength + 1 + unmatched;
            shift = max2(turboShift, bcShift);
            shift = max2(shift, m_goodSuffixShift[unmatched]);
            if (shift == m_goodSuffixShift[unmatched])
                overlap = min2(patternLength - shift, matched);
            else
            {
                if (turboShift < bcShift)
                  shift = max2(shift, overlap + 1);
                overlap = 0;
            }
        }

        index += shift;
    }
}

string S, P;
main()
{
    //freopen("testchuoi.inp", "r", stdin);
    //freopen("testchuoi.out", "w", stdin);

    cin>>S>>P;

    setString(P);
    TurboBoyerMooreMatch(S, 0);

    //fclose(stdin);
    //freopen("CON", "r", stdin);
    //system("pause");

}
```

CHAPTER V: DYNAMIC PROGRAMMING

1) CÁC BÀI TOÁN QUY HOẠCH ĐỘNG ĐIỂN HÌNH

Chúng ta đều biết rằng điều khó nhất để giải một bài toán quy hoạch động (QHĐ) là biết rằng nó là một bài toán QHĐ và tìm được công thức QHĐ của nó. Rất khó nếu ta mò mẫm từ đầu, nhưng nếu chúng ta đưa được bài toán cần giải về một bài toán QHĐ kinh điển thì sẽ dễ dàng hơn nhiều. Do đó, ***tìm hiểu mô hình, công thức và cách cài đặt những bài toán QHĐ kinh điển là một việc rất cần thiết.***

Trong chuyên đề này, tôi xin giới thiệu một số bài toán QHĐ kinh điển và những biến thể của chúng.*Chủ yếu tập trung vào giới thiệu mô hình, công thức và một số gợi ý trong cài đặt* chứ không đi chi tiết vào việc phát biểu bài toán, mô tả input/output, chứng minh công thức hay viết chương trình cụ thể. Mặc dù rất muốn minh hoạ cho các bài toán bằng các hình vẽ trực quan nhưng khuôn khổ có hạn nên tôi không thể đưa vào. *Hơn nữa phần gợi ý cài đặt chỉ có gợi ý cho phần tính bảng phương án, phần lần vết cần có các cấu trúc dữ liệu và những kĩ thuật xử lí phức tạp xin dành lại cho các bạn*.

a) Dãy con đơn điệu dài nhất

1. Mô hình

Cho dãy $a_1,a_2,..a_n$. Hãy tìm một dãy con tăng có nhiều phần tử nhất của dãy.

Đặc trưng: i) Các phần tử trong dãy kết quả chỉ xuất hiện 1 lần. Vì vậy phương pháp làm là ta sẽ dùng vòng For duyệt qua các phần tử a_i trong dãy, khác với các bài toán của mô hình 4(đặc trưng là bài toán đổi tiền), các phần tử trong dãy có thể được chọn nhiều lần nên ta thực hiện bằng phương pháp cho giá trị cần quy đổi tăng dần từng đơn vị.

ii) Thứ tự của các phần tử được chọn phải được giữ nguyên so với dãy ban đầu. Đặc trưng này có thể mất đi trong một số bài toán khác tùy vào yêu cầu cụ thể. Chẳng hạn bài Tam giác bao nhau.

2. Công thức QHĐ

Hàm mục tiêu : f = độ dài dãy con.

Vì độ dài dãy con chỉ phụ thuộc vào 1 yếu tố là dãy ban đầu nên bảng phương án là bảng một chiều. Gọi L(i) là *độ dài dãy con tăng dài nhất, các phần tử lấy trong miền từ a_1 đến a_i và phần tử cuối cùng là a_i.*

Nhận xét với cách làm này ta đã chia 1 bài toán lớn (dãy con của n số) thành các bài toán con cùng kiểu có kích thước nhỏ hơn (dãy con của dãy i số). Vấn đề là công thức truy hồi để phối hợp kết quả của các bài toán con.

Ta có công thức QHĐ để tính L(i) như sau:

- L(1) = 1. (Hiển nhiên)
- L(i) = max(1, L(j)+1 với mọi phần tử j: $0<j<i$ và $a_j \le a_i$).

Tính L(i) : phần tử đang được xét là a_i .Ta tìm đến phần tử a_j $<a_i$ có L(j) lớn nhất. Khi đó nếu bổ sung a_i vào sau dãy con ...a_j ta sẽ được dãy con tăng dần dài nhất xét từ a_1...a_i.

3. Cài đặt

Bảng phương án là một mảng một chiều L để lưu trữ các giá trị của hàm QHĐ L(i). Đoạn chương trình tính các giá trị của mảng L như sau:

```
for i := 1 to n do begin
  L[i] := 1;
  for j:=1 to i–1 do
```

```
    if (a[j]<=a[i]) and (L[i]<L[j]+1) then
                                                    L[i]:=L[j]+1;
end;
```

Như vậy chi phí không gian của bài toán là O(n), chi phí thời gian là $O(n^2)$. Có một phương pháp cài đặt tốt hơn so với phương pháp trên, cho chi phí thời gian là O(nlogn), bạn đọc có thể tham khảo trong bài báo của thầy Trần Đỗ Hùng trên tạp chí THNT số tháng 10 năm 2004.

b) Longest Increasing Subsequence Code O(NlogN)

```
// Given a list of numbers of length n, this routine extracts a
// longest increasing subsequence.
//
// Running time: O(n log n)
//
//   INPUT: a vector of integers
//   OUTPUT: a vector containing the longest increasing subsequence

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> VI;
typedef pair<int,int> PII;
typedef vector<PII> VPII;

#define STRICTLY_INCREASNG

VI LongestIncreasingSubsequence(VI v) {
  VPII best;
  VI dad(v.size(), -1);

  for (int i = 0; i < v.size(); i++) {
#ifdef STRICTLY_INCREASNG
    PII item = make_pair(v[i], 0);
    VPII::iterator it = lower_bound(best.begin(), best.end(), item);
    item.second = i;
#else
    PII item = make_pair(v[i], i);
    VPII::iterator it = upper_bound(best.begin(), best.end(), item);
#endif
    if (it == best.end()) {
      dad[i] = (best.size() == 0 ? -1 : best.back().second);
      best.push_back(item);
    } else {
      dad[i] = dad[it->second];
      *it = item;
    }
  }

  VI ret;
  for (int i = best.back().second; i >= 0; i = dad[i])
    ret.push_back(v[i]);
  reverse(ret.begin(), ret.end());
  return ret;
}
```

4. Một số bài toán khác

Bài toán dãy con đơn điệu tăng dài nhất có biến thể đơn giản nhất là bài toán dãy con đơn điệu giảm dài nhất, tuy nhiên chúng ta có thể coi chúng như là một. Sau đây là một số bài toán khác.

a) Bố trí phòng họp(mất tính thứ tự so với dãy ban đầu)

Có n cuộc họp, cuộc họp thứ i bắt đầu vào thời điểm a_i và kết thúc ở thời điểm b_i. Do chỉ có một phòng hội thảo nên 2 cuộc họp bất kì sẽ được cùng bố trí phục vụ nếu khoảng thời gian làm việc của chúng chỉ giao nhau tại đầu mút. Hãy bố trí phòng họp để phục vụ được nhiều cuộc họp nhất.

Hướng dẫn: Sắp xếp các cuộc họp tăng dần theo thời điểm kết thúc (b_i). Thế thì cuộc họp i sẽ bố trí được sau cuộc họp j nếu và chỉ nếu j<i và b_j<=a_i. Yêu cầu bố trí được nhiều cuộc họp nhất có thể đưa về việc tìm dãy các cuộc họp dài nhất thoả mãn điều kiện trên.

b) Cho thuê máy

Trung tâm tính toán hiệu năng cao nhận được đơn đặt hàng của n khách hàng. Khách hàng i muốn sử dụng máy trong khoảng thời gian từ a_i đến b_i và trả tiền thuê là c_i. Hãy bố trí lịch thuê máy để tổng số tiền thu được là lớn nhất mà thời gian sử dụng máy của 2 khách hàng bất kì được phục vụ đều không giao nhau (cả trung tâm chỉ có một máy cho thuê).

Hướng dẫn: Tương tự như bài toán a), nếu sắp xếp các đơn đặt hàng theo thời điểm kết thúc, ta sẽ đưa được bài toán b) về bài toán ***tìm dãy con có tổng lớn nhất***. Bài toán này là biến thể của bài toán tìm dãy con tăng dài nhất, ta có thể cài đặt bằng đoạn chương trình như sau:

```
for i:=1 to n do begin
   L[i]:=c[i];
   for j:=1 to i–1 do
      if (b[j]<=a[i]) and (L[i]<L[j]+c[i]) then
         L[i]:=L[j]+c[i];
end;
```

c) Dãy tam giác bao nhau

Cho n tam giác trên mặt phẳng. Tam giác i bao tam giác j nếu 3 đỉnh của tam giác j đều nằm trong tam giác i (có thể nằm trên cạnh). Hãy tìm dãy tam giác bao nhau có nhiều tam giác nhất.

Hướng dẫn: Sắp xếp các tam giác tăng dần về diện tích. Khi đó tam giác i sẽ bao tam giác j nếu j<i và 3 đỉnh của j nằm trong i. Từ đó có thể đưa về bài toán tìm dãy "tăng" dài nhất.

Việc kiểm tra điểm M có nằm trong tam giác ABC không có thể dựa trên phương pháp tính diện tích: điểm M nằm trong nếu S(ABC) = S(ABM) + S(ACM) + S(BCM).

Bài toán có một số biến thể khác như tìm dãy hình tam giác, hình chữ nhật… bao nhau có tổng diện tích lớn nhất.

d) Dãy đổi dấu

Cho dãy a_1, a_2,…a_n. Hãy dãy con đổi dấu dài nhất của dãy đó. Dãy con con đổi dấu a_{i1},a_{i2},…a_{ik} phải thoả mãn các điều kiện sau:

- a_{i1}<a_{i2}>a_{i3}<… hoặc a_{i1}>a_{i2}<a_{i3}>…
- các chỉ số phải cách nhau ít nhất L: i_2–i_1≥L, i_3–i_2≥L….
- chênh lệch giữa 2 phần tử liên tiếp nhỏ hơn U: |a_{i1}–a_{i2}|≤U, |a_{i2}–a_{i3}|≤U…

Hướng dẫn: Gọi L(i) là số phần tử của dãy con đổi dấu có phần tử cuối cùng là a_i và phần tử cuối cùng **lớn hơn** phần tử đứng trước. Tương tự, P(i) là số phần tử của dãy con đổi dấu có phần tử cuối cùng là a_i và phần tử cuối cùng **nhỏ hơn** phần tử đứng trước.

Ta dễ dàng suy ra:

- L(i) = max(1, P(j)+1): j≤i–L và a_i–U≤a_j<a_i.

- P(i) = max(1, L(j)+1): j≤i–L và $a_i<a_j≤a_i+U$.

f) Dãy số WAVIO:

Dãy số Wavio là dãy số nguyên thỏa mãn các tính chất : các phần tử đầu sắp xếp thành 1 dãy tăng dần đến 1 phần tử đỉnh sau đó giảm dần. Ví dụ dãy số *1 2 3 4* **5** *2 1* là 1 dãy Wavio độ dài 7. Cho 1 dãy gồm N số nguyên, hãy chỉ ra một dãy con Wavio có đọ dài lớn nhất trích ra từ dãy đó.

Hướng dẫn: L1[i] là mảng ghi độ dài lớn nhất của 1 dãy con tăng dần trích ra từ dãy N phần tử kể từ phần tử 1 đến phần tử a_i. L2[i] : mảng ghi độ dài lớn nhất của dãy con giảm dần trích ra từ dãy N phần tử kể từ phần tử a_N đến a_i. Ta tìm phần tử j trong 2 mảng L1, L2 thỏa mãn L1[j]+L2[j] lớn nhất.

g) Tháp Babilon *(Tính chất duy nhất của các phần tử trong phương án tối ưu bị vi phạm)*

h) Xếp các khối đá :

Cho N khối đá (N≤5000) Các khối đá đều có dạng hình hộp chữ nhật và được đặc trưng bởi 3 kích thước: dài, rộng, cao. Một cách xây dựng tháp là một cách đặt một số các khối đá trong các khối đá đã cho chồng lên nhau theo quy tắc:

- Chiều cao mỗi khối đá là kích thước nhỏ nhất trong 3 kích thước.
- Các mép của khối đá được đặt song song với nhau sao cho không có phần nào của khối trên nằm chìa ra ngoài khối dưới.

a) Hãy chỉ ra cách để xây dựng được một cái tháp sao cho số khối đá được dùng là nhiều nhất.
b) Hãy chỉ ra cách để xây dựng được một cái tháp sao cho chiều cao của cái tháp là cao nhất
Dữ liệu vào TOWER.INP có cấu trúc như sau :

- Dòng đầu là số N.
- N dòng sau dòng i ghi 3 số nguyên ≤ 255 là 3 kích thước của khối đá i . Dữ liệu ra :

TOWER1.OUT, TOWER2.OUT ghi theo quy cách :

- Dòng đầu ghi số các khối đá được chọn theo thứ tự dùng để xây tháp từ chân lên đỉnh.
- Các dòng sau ghi các khối được chọn, mỗi khối đá ghi 4 số T, D, R, C trong đó T là số thứ tự của mỗi khối đá. D, R, C là kích thước của khối đá tương ứng.

c) Vali (B)

1. Mô hình

Có n đồ vật, vật thứ i có trọng lượng a[i] và giá trị b[i]. Hãy chọn ra một số các đồ vật, mỗi vật một cái để xếp vào 1 vali có trọng lượng tối đa W sao cho tổng giá trị của vali là lớn nhất.

2. Công thức

Hàm mục tiêu : f: tổng giá trị của vali.

Nhận xét : giá trị của vali phụ thuộc vào 2 yếu tố: có bao nhiêu vật đang được xét và trọng lượng của các vật. Do đó bảng phương án sẽ là bảng 2 chiều.

L[i,j] : tổng giá trị lớn nhất của vali khi xét từ vật 1..vật i và trọng lượng của vali chưa vượt quá j. Chú ý rằng khi xét đến L[i,j] thì các giá trị trên bảng phương án đều đã được tối ưu.

- Tính L[i,j] : vật đang xét là a_i với trọng lượng của vali không được quá j. Có 2 khả năng xảy ra:

- Nếu chọn a_i đưa vào vali, trọng lượng vali trước đó phải ≤ j-a[i]. Vì mỗi vật chỉ được chọn 1 lần nên giá trị lớn nhất của vali lúc đó là L[i-1,j-a[i]) + b[i]

- Nếu không chọn a_i , trọng lượng của vali là như cũ (như lúc trước khi chọn a_i): L[i-1,j]. Tóm lại ta có L[i,j]=max(L(i-1,j-a[i]) + b[i], L[i-1,j]).

3. Cài đặt

For i:=1 to n do

```
For j:=1 to W do
        If b[i]<=j then
          L[i,j]:=max(L(i-1,j-a[i]) + b[i], L[i-1,j])
          else L[i,j]:=L[i-1,j];
```

4. Một số bài toán khác

a) Dãy con có tổng bằng S:

Cho dãy $a_1,a_2,..a_n$. Tìm một dãy con của dãy đó có tổng bằng S.

Hướng dẫn

Đặt L[i,t)=1 nếu *có thể tạo ra tổng t từ một dãy con của dãy gồm các phần tử* $a_1,a_2,..a_i$. Ngược lại thì L[i,t)=0. Nếu L[n,S)=1 thì đáp án của bài toán trên là "có".

Ta có thể tính L[i,t] theo công thức: L[i,t]=1 nếu L[i–1,t]=1 hoặc L[i–1,t–a[i]]=1.

Cài đặt

Nếu áp dụng luôn công thức trên thì ta cần dùng bảng phương án hai chiều. Ta có thể nhận xét rằng để tính dòng thứ i, ta chỉ cần dòng i–1. Bảng phương án khi đó chỉ cần 1 mảng 1 chiều L[0..S] và được tính như sau:

```
L[t]:=0; L[0]:=1;
for i := 1 to n do
   for t := S downto a[i] do
      if (L[t]=0) and (L[t–a[i]]=1) then L[t]:=1;
```

Dễ thấy chi phí không gian của cách cài đặt trên là O(m), chi phí thời gian là O(nm), với m là tổng của n số. Hãy tự kiểm tra xem tại sao vòng for thứ 2 lại là for downto chứ không phải là for to.

b) Chia kẹo

Cho n gói kẹo, gói thứ i có a_i viên. Hãy chia các gói thành 2 phần sao cho chênh lệch giữa 2 phần là ít nhất.

Hướng dẫn: Gọi T là tổng số kẹo của n gói. Chúng ta cần tìm số S **lớn nhất** thoả mãn:

- $S \le T/2$.
- Có một dãy con của dãy a có tổng bằng S.

Khi đó sẽ có cách chia với chênh lệch 2 phần là **T–2S** là nhỏ nhất và dãy con có tổng bằng S ở trên gồm các phần tử là các gói kẹo thuộc phần thứ nhất. Phần thứ hai là các gói kẹo còn lại.

c) Market (Olympic Balkan 2000)

Người đánh cá Clement bắt được n con cá, khối lượng mỗi con là a_i, đem bán ngoài chợ. Ở chợ cá, người ta không mua cá theo từng con mà mua theo một **lượng** nào đó. Chẳng hạn 3 kg, 5kg…

Ví dụ: có 3 con cá, khối lượng lần lượt là: 3, 2, 4. Mua lượng 6 kg sẽ phải lấy con cá thứ 2 và và thứ 3. Mua lượng 3 kg thì lấy con thứ nhất. Không thể mua lượng 8 kg.

Nếu bạn là người đầu tiên mua cá, có bao nhiêu lượng bạn có thể chọn?

Hướng dẫn: Thực chất bài toán là tìm các số S mà có một dãy con của dãy a có tổng bằng S. Ta có thể dùng phương pháp đánh dấu của bài chia kẹo ở trên rồi đếm các giá trị t mà L[t]=1.

d) Điền dấu

Cho n số tự nhiên $a_1,a_2, ...,a_n$. Ban đầu các số được đặt liên tiếp theo đúng thứ tự cách nhau bởi dấu "?": $a_1?a_2?...?a_n$. Cho trước số nguyên S, có cách nào thay các dấu "?" bằng dấu + hay dấu − để được một biểu thức số học cho giá trị là S không?

Hướng dẫn: Đặt L(i,t)=1 nếu có thể điền dấu vào i số đầu tiên và cho kết quả bằng t. Ta có công

thức sau để tính L:

- L(1,a[1]) =1.
- L(i,t)=1 nếu L(i–1,t+a[i])=1 hoặc L(i–1,t–a[i])=1.

Nếu L(n,S)=1 thì câu trả lời của bài toán là có. Khi cài đặt, có thể dùng một mảng 2 chiều (lưu toàn bộ bảng phương án) hoặc 2 mảng một chiều (để lưu dòng i và dòng i–1). Chú ý là chỉ số theo t của các mảng phải có cả phần âm (tức là từ –T đến T, với T là tổng của n số), vì trong bài này chúng ta dùng cả dấu – nên có thể tạo ra các tổng âm.

Bài này có một biến thể là *đặt dấu sao cho kết quả là một số chia hết cho k*. Ta có thuật giải tương tự bài toán trên bằng cách thay các phép cộng, trừ bằng các *phép cộng và trừ theo môđun k* và dùng mảng đánh dấu với các giá trị từ 0 đến k–1 (là các số dư có thể có khi chia cho k). Đáp số của bài toán là L(n,0).

e) Expression (ACM 10690)

Cho n số nguyên. Hãy chia chúng thành 2 nhóm sao cho tích của tổng 2 nhóm là lớn nhất.

Hướng dẫn: Gọi T là tổng n số nguyên đó. Giả sử ta chia dãy thành 2 nhóm, gọi S là tổng của một nhóm, tổng nhóm còn lại là T–S và tích của tổng 2 nhóm là S*(T–S). Bằng phương pháp đánh dấu ta xác định được mọi số S là tổng của một nhóm (như bài Market) và tìm số S sao cho S*(T–S) đạt max.

d) Biến đổi xâu:

1. Mô hình

Cho 2 xâu X,F. Xâu nguồn có n kí tự $X_1X_2...X_n$, xâu đích có m kí tự $F_1F_2...F_m$ .Có 3 phép biến đổi :

- Chèn 1 kí tự vào sau kí tự thứ i :I i C
- Thay thế kí tự ở vị trí thứ i bằng kí tự C : R i C.
- Xoá kí tự ở vị trí thứ i. D i

Hãy tìm số ít nhất các phép biến đổi để biến xâu X thành xâu F.

Hướng dẫn:

Hàm mục tiêu : f: số phép biến đổi.

Dễ thấy số phép biến đổi phụ thuộc vào vị trí i đang xét của xâu X và vị trí j đang xét cuả xâu F. Do vậy để cài đặt cho bang phương án ta sẽ dùng mảng 2 chiều

Gọi L(i,j) là *số phép biến đổi ít nhất để biến xâu X(i) gồm i kí tự phần đầu của X* (X(i)= X[1..i]) thành *xâu F(j) gồm j kí tự phần đầu của F*(F(j) =F[1..j]). Dễ thấy F(0,j)=j và F(i,0)=i.

Có 2 trường hợp xảy ra: Nếu X[i]=F[j] : $X_1X_2...\mathbf{X_{i-1}}\ X_i$
$F_1F_2...\mathbf{F_{j-1}}\ X_i$

thì ta chỉ phải biến đổi xâu X(i-1) thành xâu Y(j-1). Do đó F(i,j)=F(i-1,j-1). Ngược lại, ta có 3 cách biến đổi:

Xoá kí tự X[i]: $X_1X_2...\mathbf{X_{i-1}}\ X_i$

$F_1F_2...\mathbf{F_{j-1}}\ \mathbf{Fj}$

Xâu X(i-1) thành F(j). Khi đó F(i,j)=F(i-1,j)+1.(Cộng 1 là do ta đã dùng 1 phép xóa)

Thay thế X[i] bởi F[j] : $X_1X_2...\mathbf{X_{i-1}}\ Fj$

$F_1F_2...\mathbf{F_{j-1}}\ \mathbf{Fj}$

Xâu X(i-1) thành F(j-1). Khi đó F(i,j)=F(i-1,j-1)+1.

Chèn F[j] vào X(i): $X_1X_2...\mathbf{X_{i-1}}\ \mathbf{X_i}$ Fj

$F_1F_2...\mathbf{F_{j-1}}$ **Fj**

Xâu X(i) thành Y(j-1). Khi đó F(i,j)=F(i,j-1)+1. Tổng kết lại, ta có công thức QHĐ:

- F(0,j)=j
- F(i,0)=i
- F(i,j) =F(i−1,j−1) nếu X[i] = Y[j].
- F(i,j) = min(F(i−1,j),F(i,j−1),F(i−1,j−1))+1 nếu X[i]≠Y[j].

Bài này ta có thể tiết kiệm biến hơn bằng cách dùng 2 mảng 1 chiều tính lẫn nhau và một mảng đánh dấu 2 chiều để truy vết.

4. Một số bài toán khác

a) Xâu con chung dài nhất

Cho 2 xâu X,Y. Hãy tìm xâu con của X và của Y có độ dài lớn nhất.

Công thức QHĐ

Gọi L(i,j) là *độ dài xâu con chung dài nhất của xâu X(i) gồm i kí tự phần đầu của X* (X(i)= X[1..i]) và *xâu Y(j) gồm j kí tự phần đầu của Y* (Y(j) =Y[1..j]).

Ta có công thức quy hoạch động như sau:

- L(0,j)=L(i,0)=0.
- L(i,j) = L(i−1,j−1)+1 nếu X[i] = Y[j].
- L(i,j) = max(L(i−1,j), L(i,j−1)) nếu X[i]≠Y[j].

Cài đặt

Bảng phương án là một mảng 2 chiều L[0..m,0..n] để lưu các giá trị của hàm QHĐ L(i,j).
Đoạn chương trình cài đặt công thức QHĐ trên như sau:

```
for i:=0 to m do L[i,0]:=0; for j:=0 to n do L[0,j]:=0; for i:=1 to m do
   for j:=1 to n do
      if X[i]=Y[j] then
         L[i,j]:=L[i–1,j–1]+1 else
         L[i,j]:=max(L[i–1,j],L[i,j–1]);
```

Như vậy chi phí không gian của bài toán là $O(n^2)$, chi phí thời gian là $O(n^2)$. Có một phương pháp cài đặt tốt hơn, chỉ với chi phí không gian O(n) dựa trên nhận xét sau: *để tính ô L[i,j] của bảng phương án, ta chỉ cần 3 ô L[i–1,j–1],L[i–1,j] và L[i,j–1]. Tức là để tính dòng L[i] thì chỉ cần dòng L[i–1]*. Do đó ta chỉ cần 2 mảng 1 chiều để lưu dòng vừa tính (P) và dòng đang tính (L) mà thôi. Cách cài đặt mới như sau:

```
for j:=0 to n do P[j]:=0;
for i:=1 to m do begin
   L[0] := 0;
   for j:=1 to n do
      if X[i]=Y[j] then
         L[i,j]:=P[j–1]+1
      else L[i,j]:=max(P[j], L[j–1]); P := L;
end;
```

c) Bắc cầu

Hai nước Anpha và Beta nằm ở hai bên bờ sông Omega, Anpha nằm ở bờ bắc và có M thành phố được đánh số từ 1 đến m, Beta nằm ở bờ nam và có N thành phố được đánh số từ 1 đến n (theo vị trí

từ đông sang tây). Mỗi thành phố của nước này thường có quan hệ kết nghĩa với một số thành phố của nước kia. Để tăng cường tình hữu nghị, hai nước muốn xây các cây cầu bắc qua sông, mỗi cây cầu sẽ là nhịp cầu nối 2 thành phố kết nghĩa. Với yêu cầu là các cây

cầu không được cắt nhau và mỗi thành phố chỉ là đầu cầu cho nhiều nhất là một cây cầu, hãy chỉ ra cách bắc cầu được nhiều cầu nhất.

Hướng dẫn: Gọi các thành phố của Anpha lần lượt là $a_1,a_2,\ldots a_m$; các thành phố của Beta là $b_1,b_2,\ldots b_n$. Nếu thành phố a_i và b_j kết nghĩa với nhau thì coi a_i "*bằng*" b_j. Để các cây cầu không cắt nhau, nếu ta đã chọn cặp thành phố (a_i,b_j) để xây cầu thì cặp tiếp theo phải là cặp (a_u,b_v) sao cho u>i và v>j. Như vậy *các cặp thành phố được chọn xây cầu có thể coi là một dãy con chung của hai dãy a và b*.

Bài toán của chúng ta trở thành bài toán tìm dãy con chung dài nhất, ở đây hai phần tử *"bằng"* nhau nếu chúng có quan hệ *kết nghĩa*.

d) Palindrom (IOI 2000)

Một xâu gọi là xâu đối xứng (palindrom) nếu xâu đó đọc từ trái sang phải hay từ phải sang trái đều như nhau. Cho một xâu S, hãy tìm số kí tự ít nhất cần thêm vào S để S trở thành xâu đối xứng.

Hướng dẫn: Bài toán này có một công thức QHĐ như sau:

Gọi L(i,j) là *số kí tự ít nhất cần thêm vào xâu con S[i..j] của S để xâu đó trở thành đối xứng*.
Đáp số của bài toán sẽ là L(1,n) với n là số kí tự của S. Ta có công thức sau để tính L(i,j):

- L(i,i)=0.
- L(i,j)=L(i+1,j–1) nếu S[i]=S[j]
- L(i,j)=max(L(i+1,j), L(i,j–1)) nếu S[i]≠S[j]

Bạn đọc dễ dàng có thể kiểm chứng công thức đó. Ta có thể cài đặt trực tiếp công thức đó bằng phương pháp đệ quy có nhớ. Tuy nhiên khi đó chi phí không gian là $O(n^2)$. Có một phương pháp cài đặt tiết kiệm hơn (bạn đọc có thể tham khảo ở bài báo trên của thầy Trần Đỗ Hùng), tuy nhiên phương pháp đó khá phức tạp.

Ta có thuật toán đơn giản hơn như sau:

Gọi P là *xâu đảo* của S và T là *xâu con chung dài nhất* của S và P. Khi đó các kí tự của S không thuộc T cũng là các kí tự cần thêm vào để S trở thành đối xứng. Đáp số của bài toán sẽ là n–k, với k là độ dài của T.

Ví dụ: S=**edbabcd**, xâu đảo của S là P=**dcbabde**. Xâu con chung dài nhất của S và P là T=**dbabd**. Như vậy cần thêm 2 kí tự là **e** và **c** vào để S trở thành xâu đối xứng.

e) Vali (A)

1. Mô hình

Cho n vật, vật i nặng a_i và có giá trị b_i. Hãy chọn ra một số vật để cho vào balô sao cho tổng khối lượng không vượt quá W và tổng giá trị là lớn nhất. Chú ý rằng mỗi vật có thể được chọn nhiều lần.

2. Công thức

Gọi L(i,j) là *tổng giá trị lớn nhất khi được chọn i vật từ 1 đến i cho vào balô với tổng khối lượng không vượt quá j*. L(n,W) sẽ là đáp số của bài toán (là giá trị lớn nhất có được nếu chọn n vật và tổng khối lượng không vượt quá W).

Công thức tính L(i,t) như sau:

- L(i,0)=0; L(0,j)=0.
- L(i,j)=L(i,j) nếu $t<a_i$.
- L(i,t)=max(L(i-1,j), $L(i,j-a_i)+b_i$) nếu $t \geq a_i$.

Trong đó: L(i–1,j) là giá trị có được nếu không đưa vật i vào balô, $L(i,j-a_i)+b_i$ là giá trị có được nếu chọn vật i.

3. Cài đặt

Ta có thể dùng một mảng 2 chiều để lưu bảng phương án, tuy nhiên dựa trên nhận xét rằng để tính dòng i của bảng phương án chỉ cần dòng i–1, ta chỉ cần dùng 2 mảng một chiều P và L có chỉ số từ 0 đến m để lưu 2 dòng đó. Đoạn chương trình con tính bảng phương án như sau.

```
L[t] := 0; {với mọi t}
for i := 1 to n do begin
   P:=L;
   for t := 0 to m do
      if t<a[i] then L[t]:=P[t]
      else L[t] := max(P[t],P[t–a[i]]);
end;
```

Nếu để ý kĩ bạn sẽ thấy rằng đoạn trình trên chỉ viết giống công thức QHĐ chứ chưa tối ưu. Chẳng hạn đã có lệnh gán P:=L, sau đó lại có gán L[t]:=P[t] với các giá trị t<a[i] là không cần thiết. Bạn đọc có thể tự cải tiến để chương trình tối ưu hơn.

Chi phí không gian của cách cài đặt trên là O(m) và chi phí thời gian là O(n.m).

4. Một số bài toán khác

a) Farmer (IOI 2004)

Một người có N mảnh đất và M dải đất. Các mảnh đất có thể coi là một tứ giác và các dải đất thì coi như một đường thẳng. Dọc theo các dải đất ông ta trồng các cây bách, dải đất thứ i có a_i cây bách. Ông ta cũng trồng các cây bách trên viền của các mảnh đất, mảnh đất thứ j có b_j cây bách. Cả ở trên các mảnh đất và dải đất, xen giữa 2 cây bách ông ta trồng một cây ôliu. Ông ta cho con trai được chọn các mảnh đất và dải đất tuỳ ý với điều kiện tổng số cây bách không vượt quá Q. Người con trai phải chọn thế nào để có nhiều cây ôliu (loài cây mà anh ta thích) nhất.

Hướng dẫn: Dễ thấy mảnh đất thứ i có a_i cây ôliu và dải đất thứ j có b_j–1 cây ôliu. Coi các mảnh đất và dải đất là các "đồ vật", đồ vật thứ k có khối lượng w_k và giá trị v_k (nếu k là mảnh đất i thì $w_k=v_k=a_i$, nếu k là dải đất j thì $w_k=b_j$, $v_k=b_j$–1). Ta cần chọn các "đồ vật", sao cho tổng "khối lượng" của chúng không vượt Q và tổng "giá trị" là lớn nhất. Đây chính là bài toán xếp balô đã trình bày ở trên.

b) Đổi tiền

Ở đất nước Omega người ta chỉ tiêu tiền xu. Có N loại tiền xu, loại thứ i có mệnh giá là a_i đồng. Một người khách du lịch đến Omega du lịch với số tiền M đồng. Ông ta muốn đổi số tiền đó ra tiền xu Omega để tiện tiêu dùng. Ông ta cũng muốn số đồng tiền đổi được là ít nhất (cho túi tiền đỡ nặng khi đi đây đi đó). Bạn hãy giúp ông ta tìm cách đổi tiền.

Hướng dẫn: Bài toán này khá giống bài toán xếp balô (*"khối lượng"* là mệnh giá, *"giá trị"* là 1), chỉ có một số thay đổi nhỏ: *số đồng xu mỗi loại được chọn tuỳ ý* (trong bài toán xếp balô mỗi đồ vật chỉ được chọn 1 lần) và *tổng giá trị yêu cầu là nhỏ nhất*.

Do đó ta cũng xây dựng hàm QHĐ một cách tương tự: Gọi L(i,t) là số đồng xu ít nhất nếu đổi t đồng ra i loại tiền xu (từ 1 đến i). Công thức tính L(i,t) như sau:

- L(i,0)=0;
- L(0,t)= ∞ với t>0.

- L(i,t)=L(i–1,t) nếu $t<a_i$.
- L(i,t)=min(L(i–1,t), L(i,t–a_i)+1) nếu $t \geq a_i$.

Công thức này khác công thức của bài xếp balô ở chỗ: dùng hàm **min** chứ không phải hàm **max** (vì cần tìm cách chọn ít hơn) và nếu chọn đồng xu thứ i thì L(**i**,t)=L(**i**,t–a_i)+1 (vì ta vẫn còn được chọn đồng xu thứ i đó nữa), khác với khi xếp balô là: nếu chọn đồ vật thứ i thì L(**i–1**,t–a_i)+b_i vì đồ vật i chỉ được chọn một lần.

f) Nhân ma trận

1. Mô hình

Nhân một ma trận kích thước m×n với một ma trận n×p, số phép nhân phải thực hiện là ***m.n.p***. Mặt khác phép nhân các ma trận có tính kết hợp, tức là: (A.B).C = A.(B.C)
Do đó khi tính tích nhiều ma trận, ta có thể thực hiện theo các trình tự khác nhau, mỗi trình tự tính sẽ quyết định số phép nhân cần thực hiện.

Cho N ma trận $A_1,A_2 \ldots A_n$, ma trận A_i có kích thước là $d_{i-1} \times d_i$. Hãy xác định trình tự nhân ma trận $A_1.A_2 \ldots A_n$ sao cho số phép nhân cần thực hiện là ít nhất.

2. Công thức

Gọi F(i,j) là *số phép nhân để tính tích các ma trận từ A_i đến A_j ($A_i.A_{i+1}....A_j$).*

- F(i,i)=0.
- $F(i,i+1)=d_{i-1}.d_i.d_{i+1}$
- $F(i,j) = \min(F(i,k)+F(k+1,j)+d_{i-1}.d_k.d_j$ với k=i+1,i+2,...j–1) Công thức hơi phức tạp nên tôi xin giải thích như sau:
- F(i,i)=0 là hiển nhiên.
- F(i,i+1) là số phép nhân khi nhân A_i và A_{i+1}. A_i có kích thước $d_{i-1} \times d_i$, A_{i+1} có kích thước $d_i \times d_{i+1}$, do đó $F(i,i+1)=d_{i-1}.d_i.d_{i+1}$.
- Với j>i+1 thì ta thấy có thể tính $A_i.A_{i+1}....A_j$ bằng cách chọn một vị trí k nào đó để đặt ngoặc theo trình tự:

$A_i.A_{i+1}....A_j = (A_i..A_k).(A_{k+1}..A_j)$

Ma trận kết quả của phép nhân $(A_i..A_k)$ có kích thước $d_{i-1} \times d_k$, ma trận kết quả của phép nhân $(A_{k+1}..A_j)$ có kích thước $d_k \times d_j$. Với cách đặt đó ta sẽ mất F(i,k) phép nhân để có kết quả trong dấu ngoặc thứ nhất, mất thêm F(k+1,j) phép nhân để có kết quả trong dấu ngoặc thứ hai, và cuối cùng mất $d_{i-1}.d_k.d_j$ để nhân 2 ma trận kết quả đó. Từ đó tổng số phép nhân của cách đặt đó là: $F(i,k)+F(k+1,j)+d_{i-1}.d_k.d_j$.

Ta chọn vị trí k cho số phép nhân ít nhất.

3. Cài đặt

Bảng phương án là một mảng 2 chiều F để lưu F(i,j). Chú ý khi cài đặt là để tính được F(i,j), ta phải tính F(i,k) và F(k+1,j) trước. Phương pháp đơn giản để làm điều đó là phương pháp đệ quy có nhớ.

Tuy nhiên dựa vào nhận xét là trong công thức QHĐ: ***j–i lớn hơn k–i và j–k***, ta có thể tính theo trình tự khác: ***tính các phần tử F(i,j) với j–i từ nhỏ đến lớn*** (không phải là tính các giá trị F(i,j) với i,j từ nhỏ đến lớn như vẫn làm). Với cách đó, khi tính đến F(i,j) thì ta đã có F(i,k) và F(k+1,j).

Đoạn chương trình tính bảng phương án như sau:

```
for i:=1 to n do F[i,i]:=0;
for i:=1 to n–1 do
```

```
    F[i,i+1] := d[i–1]*d[i]*d[i+1];
for m:=2 to n–1 do begin for i:=1 to n–m do begin
      j:=i+m; F[i,j]:=oo;
      for k:=i+1 to j–1 do
          F[i,j]:=min(F[i,j], F[i,k]+F[k+1,j]+d[i–1]*d[k]*d[j]);
   end;
end;
```

Với cách cài đặt trên,chi phí không gian là $O(n^2)$, chi phí thời gian là $O(n^3)$ (đây là bài toán có chi phí lớn nhất trong tất cả các bài toán QHĐ thường gặp).

4. Một số bài toán khác

a) Chia đa giác

Cho một đa giác lồi N đỉnh. Bằng các đường chéo không cắt nhau, ta có thể chia đa giác thành N–2 tam giác. Hãy xác định cách chia có tổng các đường chéo ngắn nhất.

Hướng dẫn: Để đơn giản ta coi mọi đoạn thẳng nối 2 đỉnh đều là *"đường chéo"* (nếu nối 2 đỉnh trùng nhau hoặc 2 đỉnh liên tiếp thì có độ dài bằng 0).

Gọi F(i,j) là tổng độ dài các đường chéo khi chia đa giác gồm các đỉnh từ i đến j thành các tam giác. Nếu j<i+3 thì đa giác đó có ít hơn 4 đỉnh, không cần phải chia nên F(i,j)=0. Ngược lại ta xét cách chia đa giác đó bằng cách chọn một đỉnh k nằm giữa i,j và nối i,j với k. Khi đó F(i,j)=F(i,k)+F(k,j)+d(i,k)+d(k,j); *d(i,k) là độ dài đường chéo (i,k).*

Tóm lại công thức QHĐ như sau:

- F(i,j)=0 với j<i+3.
- F(i,j)=min(F(i,k)+F(k,j)+d(i,k)+d(k,j) với k=i+1,...j–1). F(1,n) là tổng đường chéo của cách chia tối ưu.

b) Biểu thức số học (IOI 1999)

Cho biểu thức $a_1 \bullet a_2 \bullet \ldots \bullet a_n$, trong đó a_i là các số thực không âm và • là một phép toán + hoặc × cho trước. Hãy đặt các dấu ngoặc để biểu thức thu được có kết quả lớn nhất.

Hướng dẫn: Gọi F(i,j) là giá trị lớn nhất có thể có của biểu thức $a_i \bullet a_{i+1} \bullet \ldots \bullet a_j$. Dễ thấy nếu i=j thì F(i,j)=a_i, nếu j=i+1 thì F(i,j)=$a_i \bullet a_j$. Nếu j>i+1 thì có thể tính biểu thức $a_i \bullet a_{i+1} \bullet \ldots \bullet a_j$ bằng cách chia thành 2 nhóm: $(a_i \bullet a_{i+1} \bullet \ldots \bullet a_k) \bullet (a_{k+1} \bullet \ldots \bullet a_j)$, Khi đó F(i,j)=F(i,k)•F(k+1,j).

Tóm lại, công thức QHĐ là:

- F(i,i)=a_i
- F(i,i+1)=$a_i \bullet a_{i+1}$
- F(i,j)=max(F(i,k)•F(k+1,j) với k=i+1,i+2,..j–1).

(Chú là là các hạng tử của dãy đều không âm và các phép toán là + hoặc × nên F(i,k) và F(k+1,j) đạt max thì F(i,k)•F(k+1,j) cũng đạt max).

g) Ghép cặp

1.Mô hình

Có n lọ hoa sắp thẳng hàng và k bó hoa được đánh số thứ tự từ nhỏ đến lớn. Cần cắm k bó hoa trên vào n lọ sao cho hoa có số thứ tự nhỏ phải đứng trước hoa có số thứ tự lớn. Giá trị thẩm mỹ tương ứng khi cắm hoa i vào lọ thứ j là v(i,j) Hãy tìm 1 cách cắm sao cho tổng giá trị thẩm mỹ là lớn nhất. Chú ý rằng mỗi bó hoa chỉ được cắm vào 1 lọ và mỗi lọ cũng chỉ cắm được 1 bó hoa. (IOI – 1999)

2. Công thức :

Nhận xét rằng bài toán nêu trên là một bài toán ghép cặp có yêu cầu về thứ tự nên ta có thể giải quyết bằng phương pháp QHĐ.

Hàm mục tiêu : f: tổng giá trị thẩm mỹ của cách cắm.

Giá trị thẩm mỹ phụ thuộc vào các hoa và các lọ đang được xét nên ta sẽ dùng mảng 2 chiều để lưu bảng phương án.

L(i,j): tổng giá trị thẩm mỹ lớn nhất khi xét đến hoa i và lọ j. Khi tính L(i,j) hoa đang xét sẽ là hoa i và lọ j.

- Nếu i = j. Chỉ có một cách cắm L(i,i):= v[1,1]+v[2,2]+...v[i,i]
- Nếu i>j . Không có cách cắm hợp lý
- Nếu i<j : Có 2 trường hợp xảy ra:

Cắm hoa i vào lọ j. Tổng giá trị thẩm mỹ là L(i-1,j-1)+v(i,j). (Bằng tổng giá trị trước khi cắm cộng với giá trị thẩm mỹ khi cắm hoa i vào lọ j)

Không cắm hoa i vào lọ j (có thể cắm vào lọ trước j), giá trị thẫm mỹ của cách cắm là như cũ : L(i,j-1)

3. Cài đặt :

```
L[i,j]:= -maxint; For i:=1 to k do
        For j:=i to n do
            If i = j then L[i,j]:=sum(i)
        else
        if i<j then L[i,j]:=max(L[i-1,j-1]+v[i,j],L[i,j-1]);
```

4. Một số bài toán khác

a) Câu lạc bộ:(Đề thi chọn HSG Tin Hà Nội năm 2000)

Có n phòng học chuyên đề và k nhóm học được đánh số thứ tự từ nhỏ đến lớn. Cần xếp k nhóm trên vào n phòng học sao cho nhóm có số hiệu nhỏ được xếp vào phòng có số hiệu nhỏ, nhóm có số hiệu lớn phải được xếp vào phòng có số hiệu lớn.Với mỗi phòng có chứ học sinh, các ghế thừa phải được chuyển ra hết, nếu thiếu ghế thì lấy vào cho đủ ghế .Biết phòng i có a(i) ghế ,nhóm j có b(j) học sinh. Hãy chọn 1 phương án bố trí sao cho tổng số lần chuyển ghế ra và vào là ít nhất.

Hướng dẫn : Khi xếp nhóm i vào phòng j thì số lần chuyển ghế chính là độ chênh lệch giữa số ghế trong phòng i và số học sinh trong nhóm. Đặt v[i,j]:=|a(i)-b(j)|

b) Mua giày (Đề QG bảng B năm 2003)

Trong hiệu có n đôi giày, đôi giày i có kích thước h_i. Có k người cần mua giày, người i cần mua đôi giày kích thước s_i. Khi người i chọn mua đôi giày j thì độ lệch sẽ là |h(i)-s(j)|. Hãy tìm cách chọn mua giày cho k người trên sao cho tổng độ lệch là ít nhất. Biết rằng mỗi người chỉ mua 1 đôi giày và 1 đôi giày cũng chỉ có một người mua.

Hướng dẫn: Lập công thức giải như bài Câu lạc bộ. Chú ý chứng minh tính đúng đắn của bổ đề heuristic sau :Cho 2 dãy tăng dần các số dương a_1 a_2...a_n , b_1 b_2...b_n. Gọi c_1 c_2...c_n là một hoán vị của dãy $\{b_n\}$. Khi đó : |a(1)-b(1)|+ |a(2)-b(2)|+...+ |a(n)-b(n)|< |a(1)-c(1)|+ |a(2)- c(2)|+...+ |a(n)-c(n)|

2) QUY HOẠCH ĐỘNG TRẠNG THÁI

a) Trò chơi trên lưới

Ngày nay các nhà khoa học đã nghĩ ra 1 trò chơi trên ma trận rất thú vị. Thông qua đó có thể đo IQ một cách khá hiệu quả. Trò chơi được mô tả như sau:

Bạn có 1 ma trận A kích thước 8 x N trên đó gồm các số nguyên là điểm của các ô đó. Người ta sẽ yêu cầu bạn chọn 1 tập khác rỗng các ô trên ma trận này sau đó tính tổng điểm trên những ô này. Trong những ô được chọn không có

hai ô nào kề cạnh. IQ của người chơi sẽ tỉ lệ thuận với số điểm nhận được. Sherry tham gia trò chơi và đạt kết quả khá tốt.Và bây giờ Sherry muốn biết tổng điểm lớn nhất nhận được trong trò chơi này là bao nhiêu. Bạn hãy giúp sherry nhé !!!

Input

Dòng 1 là số nguyên N (1 <= N <= 10000)

8 dòng tiếp theo: Mỗi dòng gồm n số nguyên. Số nguyên ở hàng i, cột j là A_{ij} ($|A_{ij}| <= 10^8$)

Output

Gồm 1 dòng duy nhất là số điểm lớn nhất tìm được

Example

Input:

2
-22 2
-33 45
56 -60
-8 -38
79 66
-10 -23
99 46
1 -55

Output:

279

Giải thích:

Chọn các ô (3,1) (5,1) (7,1) (2,2)

Lời giải:

Xét bảng 4*N, gọi F[i,j] là giá trị tối ưu khi xét đến cột i và j là số có 4 bit để miêu tả trạng thái của cột i(bit thứ k của số j =1 nếu như chọn ô k).

F[i,j]=max (F[i-1,j'] + sum(i,j)) trong đó j' là cấu hình của cột thứ i-1 và j' và j thõa mãn với nhau, sum(i,x) là tổng của các số trong cột i tương đương với số j

3) MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG BỔ XUNG

Cho dãy số gồm n số nguyên $a_0, a_1, \ldots, a_{n-1}$. Giá trị tối ưu trên dãy số được tính bằng hàm f() như sau:

$f(i, j, k, z, l)=a_i + 2*a_j + 3*a_k + 4*a_z + 5*a_l$ với $0 \le i < j < k < z < l < n$

```
#include<stdio.h>
#include<iostream>
#include<iomanip.h>
using namespace std;
int n;
int hs[6]={0,1,2,3,4,5};
int a[10002];
int f[6][10002];
main(){
   int i,sotest;
   //freopen("gttoiuu.inp","r",stdin);
   //freopen("timdaymaxout.txt","w",stdout);
   cin>>sotest;
   for(i=0;i<sotest;i++)
   {
      cin>>n;
      for(int i=1;i<=n;i++)
         cin>>a[i];
      //thiet lap co so quy hoach dong
      for(int i=0;i<=n;i++)
      {
         f[0][i]=0;
```

```
        }
        for(int i=1;i<6;i++)
           f[i][0]=0;

        //Quy hoach dong
        for(int i=1;i<6;i++)
        {
           f[i][i]=f[i-1][i-1]+a[i]*hs[i];
           for(int j=i+1;j<=n-5+i;j++)
           {
              f[i][j]=f[i][j-1];
              if(f[i-1][j-1] + a[j]*hs[i]>f[i][j])
                 f[i][j]=f[i-1][j-1] + a[j]*hs[i];
           }
        }
        cout<<f[5][n]<<endl;
        //truy vet
/*
        int i=5;
        int j=n;
        for(;j>0;j--)
           if(f[i][j]!=f[i][j-1]){
        cout<<j-1<<" ";
        i--;
        if(i==0)break;
        }
        cout<<endl;
*/
    }
}
```

CHAPTER VI: GEOMETRY

1) Convex Hull

```
// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm.  Eliminate redundant points from the hull if REMOVE_REDUNDANT is
// #defined.
//
// Running time: O(n log n)
//
//   INPUT:   a vector of input points, unordered.
//   OUTPUT:  a vector of points in the convex hull, counterclockwise, starting
//            with bottommost/leftmost point

#include <cstdio>
#include <cassert>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs) const { return make_pair(y,x) <
make_pair(rhs.y,rhs.x); }
  bool operator==(const PT &rhs) const { return make_pair(y,x) ==
make_pair(rhs.y,rhs.x); }
};

T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) + cross(c,a); }

#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
  return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <= 0 && (a.y-b.y)*(c.y-
b.y) <= 0);
}
#endif

void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(), pts.end()), pts.end());
  vector<PT> up, dn;
  for (int i = 0; i < pts.size(); i++) {
    while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0)
up.pop_back();
    while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0)
dn.pop_back();
    up.push_back(pts[i]);
    dn.push_back(pts[i]);
  }
  pts = dn;
  for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT
  if (pts.size() <= 2) return;
  dn.clear();
  dn.push_back(pts[0]);
```

```
  dn.push_back(pts[1]);
  for (int i = 2; i < pts.size(); i++) {
    if (between(dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
    dn.push_back(pts[i]);
  }
  if (dn.size() >= 3 && between(dn.back(), dn[0], dn[1])) {
    dn[0] = dn.back();
    dn.pop_back();
  }
  pts = dn;
#endif
}
```

2) Geometry Computation

```
// C++ routines for computational geometry.

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y)    {}
  PT operator + (const PT &p)  const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p)  const { return PT(x-p.x, y-p.y); }
  PT operator * (double c)     const { return PT(x*c,   y*c  ); }
  PT operator / (double c)     const { return PT(x/c,   y/c  ); }
};

double dot(PT p, PT q)     { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q)   { return dot(p-q,p-q); }
double cross(PT p, PT q)   { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p)   { return PT(-p.y,p.x); }
PT RotateCW90(PT p)    { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
```

```
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d) && fabs(cross(a-c, d-c)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
```

```
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r+R || d+min(r, R) < max(r, R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back(a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back(a+v*x - RotateCCW90(v)*y);
  return ret;
}

// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion.  Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}
```

```
PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}

int main() {

  // expected: (-5,2)
  cerr << RotateCCW90(PT(2,5)) << endl;

  // expected: (5,-2)
  cerr << RotateCW90(PT(2,5)) << endl;

  // expected: (-5,2)
  cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

  // expected: (5,2)
  cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

  // expected: (5,2) (7.5,3) (2.5,1)
  cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
       << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
       << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

  // expected: 6.78903
  cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

  // expected: 1 0 1
  cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 0 0 1
  cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

  // expected: 1 1 1 0
  cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

  // expected: (1,2)
  cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << endl;
```

```
  // expected: (1,1)
  cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;

  vector<PT> v;
  v.push_back(PT(0,0));
  v.push_back(PT(5,0));
  v.push_back(PT(5,5));
  v.push_back(PT(0,5));

  // expected: 1 1 1 0 0
  cerr << PointInPolygon(v, PT(2,2)) << " "
       << PointInPolygon(v, PT(2,0)) << " "
       << PointInPolygon(v, PT(0,2)) << " "
       << PointInPolygon(v, PT(5,2)) << " "
       << PointInPolygon(v, PT(2,5)) << endl;

  // expected: 0 1 1 1 1
  cerr << PointOnPolygon(v, PT(2,2)) << " "
       << PointOnPolygon(v, PT(2,0)) << " "
       << PointOnPolygon(v, PT(0,2)) << " "
       << PointOnPolygon(v, PT(5,2)) << " "
       << PointOnPolygon(v, PT(2,5)) << endl;

  // expected: (1,6)
  //           (5,4) (4,5)
  //           blank line
  //           (4,5) (5,4)
  //           blank line
  //           (4,5) (5,4)
  vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;

  // area should be 5.0
  // centroid should be (1.1666666, 1.166666)
  PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
  vector<PT> p(pa, pa+4);
  PT c = ComputeCentroid(p);
  cerr << "Area: " << ComputeArea(p) << endl;
  cerr << "Centroid: " << c << endl;

  return 0;
}
```

3) JavaGeometry

```
// In this example, we read an input file containing three lines, each
// containing an even number of doubles, separated by commas.  The first two
// lines represent the coordinates of two polygons, given in counterclockwise
// (or clockwise) order, which we will call "A" and "B".  The last line
// contains a list of points, p[1], p[2], ...
//
// Our goal is to determine:
//   (1) whether B - A is a single closed shape (as opposed to multiple shapes)
//   (2) the area of B - A
//   (3) whether each p[i] is in the interior of B - A
//
// INPUT:
//   0 0 10 0 0 10
//   0 0 10 10 10 0
//   8 6
//   5 1
//
// OUTPUT:
//   The area is singular.
//   The area is 25.0
//   Point belongs to the area.
//   Point does not belong to the area.

import java.util.*;
import java.awt.geom.*;
import java.io.*;

public class JavaGeometry {

    // make an array of doubles from a string
    static double[] readPoints(String s) {
        String[] arr = s.trim().split("\\s++");
        double[] ret = new double[arr.length];
        for (int i = 0; i < arr.length; i++) ret[i] = Double.parseDouble(arr[i]);
        return ret;
    }

    // make an Area object from the coordinates of a polygon
    static Area makeArea(double[] pts) {
        Path2D.Double p = new Path2D.Double();
        p.moveTo(pts[0], pts[1]);
        for (int i = 2; i < pts.length; i += 2) p.lineTo(pts[i], pts[i+1]);
        p.closePath();
        return new Area(p);
    }

    // compute area of polygon
    static double computePolygonArea(ArrayList<Point2D.Double> points) {
        Point2D.Double[] pts = points.toArray(new Point2D.Double[points.size()]);
        double area = 0;
        for (int i = 0; i < pts.length; i++){
            int j = (i+1) % pts.length;
            area += pts[i].x * pts[j].y - pts[j].x * pts[i].y;
        }
        return Math.abs(area)/2;
    }

    // compute the area of an Area object containing several disjoint polygons
    static double computeArea(Area area) {
        double totArea = 0;
        PathIterator iter = area.getPathIterator(null);
        ArrayList<Point2D.Double> points = new ArrayList<Point2D.Double>();
```

```
        while (!iter.isDone()) {
            double[] buffer = new double[6];
            switch (iter.currentSegment(buffer)) {
            case PathIterator.SEG_MOVETO:
            case PathIterator.SEG_LINETO:
                points.add(new Point2D.Double(buffer[0], buffer[1]));
                break;
            case PathIterator.SEG_CLOSE:
                totArea += computePolygonArea(points);
                points.clear();
                break;
            }
            iter.next();
        }
        return totArea;
    }

    // notice that the main() throws an Exception -- necessary to
    // avoid wrapping the Scanner object for file reading in a
    // try { ... } catch block.
    public static void main(String args[]) throws Exception {

        Scanner scanner = new Scanner(new File("input.txt"));
        // also,
        //   Scanner scanner = new Scanner (System.in);

        double[] pointsA = readPoints(scanner.nextLine());
        double[] pointsB = readPoints(scanner.nextLine());
        Area areaA = makeArea(pointsA);
        Area areaB = makeArea(pointsB);
        areaB.subtract(areaA);
        // also,
        //   areaB.exclusiveOr (areaA);
        //   areaB.add (areaA);
        //   areaB.intersect (areaA);

        // (1) determine whether B - A is a single closed shape (as
        //     opposed to multiple shapes)
        boolean isSingle = areaB.isSingular();
        // also,
        //   areaB.isEmpty();

        if (isSingle)
            System.out.println("The area is singular.");
        else
            System.out.println("The area is not singular.");

        // (2) compute the area of B - A
        System.out.println("The area is " + computeArea(areaB) + ".");

        // (3) determine whether each p[i] is in the interior of B - A
        while (scanner.hasNextDouble()) {
            double x = scanner.nextDouble();
            assert(scanner.hasNextDouble());
            double y = scanner.nextDouble();

            if (areaB.contains(x,y)) {
                System.out.println ("Point belongs to the area.");
            } else {
                System.out.println ("Point does not belong to the area.");
            }
        }

        // Finally, some useful things we didn't use in this example:
        //
```

```
        //   Ellipse2D.Double ellipse = new Ellipse2D.Double (double x, double y,
        //                                                    double w, double h);
        //
        //     creates an ellipse inscribed in box with bottom-left corner (x,y)
        //     and upper-right corner (x+y,w+h)
        //
        //   Rectangle2D.Double rect = new Rectangle2D.Double (double x, double y,
        //                                                     double w, double
h);
        //
        //     creates a box with bottom-left corner (x,y) and upper-right
        //     corner (x+y,w+h)
        //
        // Each of these can be embedded in an Area object (e.g., new Area
(rect)).

    }
}
```

4) Geometry 3D

```
public class Geom3D {
  // distance from point (x, y, z) to plane aX + bY + cZ + d = 0
  public static double ptPlaneDist(double x, double y, double z,
      double a, double b, double c, double d) {
    return Math.abs(a*x + b*y + c*z + d) / Math.sqrt(a*a + b*b + c*c);
  }

  // distance between parallel planes aX + bY + cZ + d1 = 0 and
  // aX + bY + cZ + d2 = 0
  public static double planePlaneDist(double a, double b, double c,
      double d1, double d2) {
    return Math.abs(d1 - d2) / Math.sqrt(a*a + b*b + c*c);
  }

  // distance from point (px, py, pz) to line (x1, y1, z1)-(x2, y2, z2)
  // (or ray, or segment; in the case of the ray, the endpoint is the
  // first point)
  public static final int LINE = 0;
  public static final int SEGMENT = 1;
  public static final int RAY = 2;
  public static double ptLineDistSq(double x1, double y1, double z1,
      double x2, double y2, double z2, double px, double py, double pz,
      int type) {
    double pd2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) + (z1-z2)*(z1-z2);

    double x, y, z;
    if (pd2 == 0) {
      x = x1;
      y = y1;
      z = z1;
    } else {
      double u = ((px-x1)*(x2-x1) + (py-y1)*(y2-y1) + (pz-z1)*(z2-z1)) / pd2;
      x = x1 + u * (x2 - x1);
      y = y1 + u * (y2 - y1);
      z = z1 + u * (z2 - z1);
      if (type != LINE && u < 0) {
        x = x1;
        y = y1;
        z = z1;
      }
      if (type == SEGMENT && u > 1.0) {
        x = x2;
        y = y2;
        z = z2;
```

```
      }
    }

    return (x-px)*(x-px) + (y-py)*(y-py) + (z-pz)*(z-pz);
  }

  public static double ptLineDist(double x1, double y1, double z1,
      double x2, double y2, double z2, double px, double py, double pz,
      int type) {
    return Math.sqrt(ptLineDistSq(x1, y1, z1, x2, y2, z2, px, py, pz, type));
  }
}
```

5) Delaunay triangulation

```
// Slow but simple Delaunay triangulation. Does not handle
// degenerate cases (from O'Rourke, Computational Geometry in C)
//
// Running time: O(n^4)
//
// INPUT:    x[] = x-coordinates
//           y[] = y-coordinates
//
// OUTPUT:   triples = a vector containing m triples of indices
//                     corresponding to triangle vertices

#include<vector>
using namespace std;

typedef double T;

struct triple {
    int i, j, k;
    triple() {}
    triple(int i, int j, int k) : i(i), j(j), k(k) {}
};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T>& y) {
        int n = x.size();
        vector<T> z(n);
        vector<triple> ret;

        for (int i = 0; i < n; i++)
            z[i] = x[i] * x[i] + y[i] * y[i];

        for (int i = 0; i < n-2; i++) {
            for (int j = i+1; j < n; j++) {
                for (int k = i+1; k < n; k++) {
                    if (j == k) continue;
                    double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j]-z[i]);
                    double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k]-z[i]);
                    double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j]-y[i]);
                    bool flag = zn < 0;
                    for (int m = 0; flag && m < n; m++)
                        flag = flag && ((x[m]-x[i])*xn +
                                        (y[m]-y[i])*yn +
                                        (z[m]-z[i])*zn <= 0);
                    if (flag) ret.push_back(triple(i, j, k));
                }
            }
        }
        return ret;
}

int main()
{
```

```
    T xs[]={0, 0, 1, 0.9};
    T ys[]={0, 1, 0, 0.9};
    vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
    vector<triple> tri = delaunayTriangulation(x, y);

    //expected: 0 1 3
    //          0 3 2

    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

CHAPTER VII: SPECIAL DATA STRUCTURE

1) Binary index tree

```
#include <iostream>
using namespace std;

#define LOGSZ 17

int tree[(1<<LOGSZ)+1];
int N = (1<<LOGSZ);

// add v to value at x
void set(int x, int v) {
  while(x <= N) {
    tree[x] += v;
    x += (x & -x);
  }
}

// get cumulative sum up to and including x
int get(int x) {
  int res = 0;
  while(x) {
    res += tree[x];
    x -= (x & -x);
  }
  return res;
}

// get largest value with cumulative sum less than or equal to x;
// for smallest, pass x-1 and add 1 to result
int getind(int x) {
  int idx = 0, mask = N;
  while(mask && idx < N) {
    int t = idx + mask;
    if(x >= tree[t]) {
      idx = t;
      x -= tree[t];
    }
    mask >>= 1;
  }
  return idx;
}
```

2) Disjoint-Set

```
//union-find set: the vector/array contains the parent of each node
int find(vector <int>& C, int x){return (C[x]==x) ? x : C[x]=find(C, C[x]);} //C++
int find(int x){return (C[x]==x)?x:C[x]=find(C[x]);} //C
```

3) KD-Tree

```
// --------------------------------------------------------------------------
// A straightforward, but probably sub-optimal KD-tree implmentation that's
// probably good enough for most things (current it's a 2D-tree)
//
//  - constructs from n points in O(n lg^2 n) time
//  - handles nearest-neighbor query in O(lg n) if points are well distributed
//  - worst case for nearest-neighbor may be linear in pathological case
//
// Sonny Chan, Stanford University, April 2009
// --------------------------------------------------------------------------

#include <iostream>
#include <vector>
```

```
#include <limits>
#include <cstdlib>

using namespace std;

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

bool operator==(const point &a, const point &b)
{
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b)
{
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b)
{
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b)
{
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox
{
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < v.size(); ++i) {
            x0 = min(x0, v[i].x);   x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y);   y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)       return pdist2(point(x0, y0), p);
            else if (p.y > y1)  return pdist2(point(x0, y1), p);
            else                return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)       return pdist2(point(x1, y0), p);
            else if (p.y > y1)  return pdist2(point(x1, y1), p);
            else                return pdist2(point(x1, p.y), p);
```

```
        }
        else {
            if (p.y < y0)       return pdist2(point(p.x, y0), p);
            else if (p.y > y1)  return pdist2(point(p.x, y1), p);
            else                return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or leaf
struct kdnode
{
    bool leaf;      // true if this is a leaf node (has one point)
    point pt;       // the single point of this is a leaf
    bbox bound;     // bounding box for set of points in children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete second; }

    // intersect a point with this node (returns squared distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp)
    {
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        }
        else {
            // split on x if the bbox is wider than high (not best heuristic...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode();   first->construct(vl);
            second = new kdnode();  second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree
{
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
```

```
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to nearest point
    ntype search(kdnode *node, const point &p)
    {
        if (node->leaf) {
            // commented special case tells a point not to find itself
//            if (p == node->pt) return sentry;
//            else
              return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        }
        else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};

// --------------------------------------------------------------------------
// some basic test code here

int main()
{
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i) {
        vp.push_back(point(rand()%100000, rand()%100000));
    }
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ", " << q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }

    return 0;
}

// --------------------------------------------------------------------------
```

4) SuffixArray

```
// Suffix array construction in O(L log^2 L) time.  Routine for
// computing the length of the longest common prefix of any two
// suffixes in O(log L) time.
//
// INPUT:   string s
//
// OUTPUT:  array suffix[] such that suffix[i] = index (from 0 to L-1)
//          of substring s[i...L-1] in the list of sorted suffixes.
//          That is, if we take the inverse of the permutation suffix[],
//          we get the actual suffix array.

#include <vector>
#include <iostream>
#include <string>

using namespace std;

struct SuffixArray {
  const int L;
  string s;
  vector<vector<int> > P;
  vector<pair<pair<int,int>,int> > M;

  SuffixArray(const string &s) : L(s.length()), s(s), P(1, vector<int>(L, 0)),
M(L) {
    for (int i = 0; i < L; i++) P[0][i] = int(s[i]);
    for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
      P.push_back(vector<int>(L, 0));
      for (int i = 0; i < L; i++)
        M[i] = make_pair(make_pair(P[level-1][i], i + skip < L ? P[level-1][i +
skip] : -1000), i);
      sort(M.begin(), M.end());
      for (int i = 0; i < L; i++)
        P[level][M[i].second] = (i > 0 && M[i].first == M[i-1].first) ?
P[level][M[i-1].second] : i;
    }
  }

  vector<int> GetSuffixArray() { return P.back(); }

  // returns the length of the longest common prefix of s[i...L-1] and s[j...L-1]
  int LongestCommonPrefix(int i, int j) {
    int len = 0;
    if (i == j) return L - i;
    for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--) {
      if (P[k][i] == P[k][j]) {
        i += 1 << k;
        j += 1 << k;
        len += 1 << k;
      }
    }
    return len;
  }
};

int main() {

  // bobocel is the 0'th suffix
  //  obocel is the 5'th suffix
  //   bocel is the 1'st suffix
  //    ocel is the 6'th suffix
  //     cel is the 2'nd suffix
  //      el is the 3'rd suffix
  //       l is the 4'th suffix
  SuffixArray suffix("bobocel");
```

```
  vector<int> v = suffix.GetSuffixArray();

  // Expected output: 0 5 1 6 2 3 4
  //                  2
  for (int i = 0; i < v.size(); i++) cout << v[i] << " ";
  cout << endl;
  cout << suffix.LongestCommonPrefix(0, 2) << endl;
}
```

5) Disjoint-Set

```
#include <iostream>
#include <stdio.h>
#include <vector>
#define MAXN 10002

using namespace std;

struct Node
{
    int rank; // This roughly represent the max height of the node in its subtree
    int index; // The index of the element the node represents
    Node* parent; // The parent node of the node
};

Node m_nodes[MAXN];

void InitElement()
{
    // insert and initialize the specified number of element nodes to the end of t
he `m_nodes` array
    for(int i = 0; i < MAXN; ++i)
    {
        m_nodes[i].parent = NULL;
        m_nodes[i].index = i;
        m_nodes[i].rank = 0;
    }
}


// Note: some internal data is modified for optimization even though this method i
s consant.
int FindSet(int x)
{
    Node* curNode;

    // Find the root element that represents the set which `x` belongs to
    curNode = m_nodes+x;
    while(curNode->parent != NULL)
        curNode = curNode->parent;
    Node* root = curNode;

    // Walk to the root, updating the parents of `elementId`. Make those elements
the direct
    // children of `root`. This optimizes the tree for future FindSet invokations.
    curNode = m_nodes+x;
    while(curNode != root)
    {
        Node* next = curNode->parent;
        curNode->parent = root;
        curNode = next;
    }

    return root->index;
}
```

```
void Union(int setId1, int setId2)
{

    if(setId1 == setId2)
        return; // already unioned

    Node* set1 = m_nodes+setId1;
    Node* set2 = m_nodes+setId2;

    if(set1->rank > set2->rank)
        set2->parent = set1;
    else if(set1->rank < set2->rank)
        set1->parent = set2;
    else // set1->rank == set2->rank
    {
        set2->parent = set1;
        ++set1->rank; // update rank
    }
}



main()
{
    //freopen("IOIBIN.INP", "r", stdin);
    //freopen("IOIBIN.OUT", "w", stdout);
    InitElement();

    int P;
    scanf("%d", &P);

    for(int ii = 1; ii<=P; ii++)
    {
        int x, y, v;
        scanf("%d %d %d", &x, &y, &v);
        if(v==1)
            Union(FindSet(x), FindSet(y));
        else
        {
            int r1 = FindSet(x);
            int r2 = FindSet(y);
            if(r1 == r2 && r1 != 0)
                printf("1\n");
            else
                printf("0\n");
        }
    }


}
```

6) Interval tree

```
#include <iostream>
#include <algorithm>

const int maxN = 50001;

using namespace std;

int k[maxN];
int iMin[5*maxN], iMax[5*maxN];
int n, q;
```

```
int thap, cao;

void initMax(int i, int l, int r)
{
    if(l==r)
        iMax[i] = k[l];
    else
    {
        int mid = (l+r)>>1;
        initMax(i*2, l, mid);
        initMax(i*2+1, mid+1, r);
        iMax[i] = max(iMax[i*2], iMax[i*2+1]);
    }
}

void initMin(int i, int l, int r)
{
    if(l==r)
        iMin[i] = k[l];
    else
    {
        int mid = (l+r)>>1;
        initMin(i*2, l, mid);
        initMin(i*2+1, mid+1, r);
        iMin[i] = min(iMin[i*2], iMin[i*2+1]);
    }
}

void findMin(int i, int l, int r, int d, int c)
{
    if(d==l && c==r) thap = min(iMin[i], thap);
    else
    {
        int mid = (l+r)>>1;
        if(d<=mid)
            findMin(i*2, l, mid, d, min(mid, c));
        if(c>mid)
            findMin(2*i+1, mid+1, r, max(mid+1, d), c);
    }
}

void findMax(int i, int l, int r, int d, int c)
{
    if(d==l && c==r) cao = max(cao, iMax[i]);
    else
    {
        int mid = (l+r)>>1;
        if(d<=mid)
            findMax(i*2, l, mid, d, min(mid, c));
        if(c>mid)
            findMax(2*i+1, mid+1, r, max(mid+1, d), c);
    }
}

main()
{
    //freopen("NKLINEUP.INP", "r", stdin);
    //freopen("NKLINEUP.OUT", "w", stdout);
    scanf("%d %d", &n, &q);
    for(int i=1; i<=n; i++)
        scanf("%d", k+i);

    initMin(1, 1, n);
```

```
    initMax(1, 1, n);

    for(int i = 0; i<q; i++)
    {
        int q1, q2;
        scanf("%d %d", &q1, &q2);
        cao=-9999999; thap=9999999;
        findMax(1, 1, n, q1, q2);
        findMin(1, 1, n, q1, q2);
        printf("%d\n", cao-thap);
    }

}
```