Sectumsempra模板——tbw部分

目录

快速傅里叶变换

```
#include <algorithm>
#include <complex>
#include <vector>
#include <cmath>
#define foreach(e,x) for(__typeof(x.begin()) e=x.begin();e!=x.end();++e)
using namespace std;

typedef complex<double> Comp;

const Comp I(0, 1);

const int MAX_N = 1 << 20;
Comp tmp[MAX_N];
void DFT(Comp*a, int n, int rev) {
    if (n == 1)
        return;
    for (int i = 0; i < n; ++i) {
        tmp[i] = a[i];
    }
    for (int i = 0; i < n; ++i) {
        if (i & 1)
            a[n / 2 + i / 2] = tmp[i];
        else
            a[i / 2] = tmp[i];
    }
    Comp*a0 = a, *a1 = a + n / 2;
    DFT(a0, n / 2, rev);
    DFT(a1, n / 2, rev);
    Comp cur(1, 0);
    double alpha = 2 * M_PI / n * rev;
    Comp step = exp(I * alpha);
    for (int k = 0; k < n / 2; ++k) {
        tmp[k] = a0[k] + cur * a1[k];
        tmp[k + n / 2] = a0[k] - cur * a1[k];
        cur *= step;
    }
    for (int i = 0; i < n; ++i) {
        a[i] = tmp[i];
    }
}

int main() {
```

```
    static Comp a[1 << 20] = { }, b[1 << 20] = { };
    int n = 1 << 20;
    DFT(a, n, 1);
    DFT(b, n, 1);
    for (int i = 0; i < n; ++i) {
        a[i] *= b[i];
    }
    DFT(a, n, -1);
    for (int i = 0; i < n; ++i) {
        a[i] /= n;
    }
}
```

线性规划

```
// UVa10498 Happiness!
// Rujia Liu
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cassert>
using namespace std;

// 改进单纯性法的实现
// 参考：http://en.wikipedia.org/wiki/Simplex_algorithm
// 输入矩阵a描述线性规划的标准形式。a为m+1行n+1列，其中行0~m-1为不等式，行m为目标函数（最大化）。列0~n-1为变量0~n-1的系数，列n为常数项
// 第i个约束为a[i][0]*x[0] + a[i][1]*x[1] + ... <= a[i][n]
// 目标为max(a[m][0]*x[0] + a[m][1]*x[1] + ... + a[m][n-1]*x[n-1] - a[m][n])
// 注意：变量均有非负约束x[i] >= 0
const int maxm = 500; // 约束数目上限
const int maxn = 500; // 变量数目上限
const double INF = 1e100;
const double eps = 1e-10;

struct Simplex {
  int n; // 变量个数
  int m; // 约束个数
  double a[maxm][maxn]; // 输入矩阵
  int B[maxm], N[maxn]; // 算法辅助变量

  void pivot(int r, int c) {
    swap(N[c], B[r]);
    a[r][c] = 1 / a[r][c];
    for(int j = 0; j <= n; j++) if(j != c) a[r][j] *= a[r][c];
    for(int i = 0; i <= m; i++) if(i != r) {
      for(int j = 0; j <= n; j++) if(j != c) a[i][j] -= a[i][c] * a[r][j];
      a[i][c] = -a[i][c] * a[r][c];
    }
  }

  bool feasible() {
    for(;;) {
      int r, c;
      double p = INF;
      for(int i = 0; i < m; i++) if(a[i][n] < p) p = a[r = i][n];
      if(p > -eps) return true;
      p = 0;
      for(int i = 0; i < n; i++) if(a[r][i] < p) p = a[r][c = i];
      if(p > -eps) return false;
      p = a[r][n] / a[r][c];
      for(int i = r+1; i < m; i++) if(a[i][c] > eps) {
        double v = a[i][n] / a[i][c];
        if(v < p) { r = i; p = v; }
      }
      pivot(r, c);
```

```
    }
  }

  // 解有界返回1，无解返回0，无界返回-1。b[i]为x[i]的值，ret为目标函数的值
  int simplex(int n, int m, double x[maxn], double& ret) {
    this->n = n;
    this->m = m;
    for(int i = 0; i < n; i++) N[i] = i;
    for(int i = 0; i < m; i++) B[i] = n+i;
    if(!feasible()) return 0;
    for(;;) {
      int r, c;
      double p = 0;
      for(int i = 0; i < n; i++) if(a[m][i] > p) p = a[m][c = i];
      if(p < eps) {
        for(int i = 0; i < n; i++) if(N[i] < n) x[N[i]] = 0;
        for(int i = 0; i < m; i++) if(B[i] < n) x[B[i]] = a[i][n];
        ret = -a[m][n];
        return 1;
      }
      p = INF;
      for(int i = 0; i < m; i++) if(a[i][c] > eps) {
        double v = a[i][n] / a[i][c];
        if(v < p) { r = i; p = v; }
      }
      if(p == INF) return -1;
      pivot(r, c);
    }
  }
};

//////////////// 题目相关
#include<cmath>
Simplex solver;

int main() {
  int n, m;
  while(scanf("%d%d", &n, &m) == 2) {
    for(int i = 0; i < n; i++) scanf("%lf", &solver.a[m][i]); // 目标函数
    solver.a[m][n] = 0; // 目标函数常数项
    for(int i = 0; i < m; i++)
      for(int j = 0; j < n+1; j++)
        scanf("%lf", &solver.a[i][j]);
    double ans, x[maxn];
    assert(solver.simplex(n, m, x, ans) == 1);
    ans *= m;
    printf("Nasa can spend %d taka.\n", (int)floor(ans + 1 - eps));
  }
  return 0;
}
```

对踵点对

```
// LA4728/UVa1453 Square
// Rujia Liu
#include<cstdio>
#include<vector>
#include<cmath>
#include<algorithm>
using namespace std;

struct Point {
  int x, y;
  Point(int x=0, int y=0):x(x),y(y) { }
};
```

```
typedef Point Vector;

Vector operator - (const Point& A, const Point& B) {
  return Vector(A.x-B.x, A.y-B.y);
}

int Cross(const Vector& A, const Vector& B) {
  return A.x*B.y - A.y*B.x;
}

int Dot(const Vector& A, const Vector& B) {
  return A.x*B.x + A.y*B.y;
}

int Dist2(const Point& A, const Point& B) {
  return (A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y);
}

bool operator < (const Point& p1, const Point& p2) {
  return p1.x < p2.x || (p1.x == p2.x && p1.y < p2.y);
}

bool operator == (const Point& p1, const Point& p2) {
  return p1.x == p2.x && p1.y == p2.y;
}

// 点集凸包
// 如果不希望在凸包的边上有输入点，把两个 <= 改成 <
// 注意：输入点集会被修改
vector<Point> ConvexHull(vector<Point>& p) {
  // 预处理，删除重复点
  sort(p.begin(), p.end());
  p.erase(unique(p.begin(), p.end()), p.end());

  int n = p.size();
  int m = 0;
  vector<Point> ch(n+1);
  for(int i = 0; i < n; i++) {
    while(m > 1 && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
    ch[m++] = p[i];
  }
  int k = m;
  for(int i = n-2; i >= 0; i--) {
    while(m > k && Cross(ch[m-1]-ch[m-2], p[i]-ch[m-2]) <= 0) m--;
    ch[m++] = p[i];
  }
  if(n > 1) m--;
  ch.resize(m);
  return ch;
}

// 返回点集直径的平方
int diameter2(vector<Point>& points) {
  vector<Point> p = ConvexHull(points);
  int n = p.size();
  if(n == 1) return 0;
  if(n == 2) return Dist2(p[0], p[1]);
  p.push_back(p[0]); // 免得取模
  int ans = 0;
  for(int u = 0, v = 1; u < n; u++) {
    // 一条直线贴住边p[u]-p[u+1]
    for(;;) {
      // 当Area(p[u], p[u+1], p[v+1]) <= Area(p[u], p[u+1], p[v])时停止旋转
      // 即Cross(p[u+1]-p[u], p[v+1]-p[u]) - Cross(p[u+1]-p[u], p[v]-p[u]) <= 0
      // 根据Cross(A,B) - Cross(A,C) = Cross(A,B-C)
      // 化简得Cross(p[u+1]-p[u], p[v+1]-p[v]) <= 0
```

```
      int diff = Cross(p[u+1]-p[u], p[v+1]-p[v]);
      if(diff <= 0) {
        ans = max(ans, Dist2(p[u], p[v])); // u和v是对踵点
        if(diff == 0) ans = max(ans, Dist2(p[u], p[v+1])); // diff == 0时u和v+1也是对踵点
        break;
      }
      v = (v + 1) % n;
    }
  }
  return ans;
}

int main() {
  int T;
  scanf("%d", &T);
  while(T--) {
    int n;
    scanf("%d", &n);
    vector<Point> points;
    for(int i = 0; i < n; i++) {
      int x, y, w;
      scanf("%d%d%d", &x, &y, &w);
      points.push_back(Point(x, y));
      points.push_back(Point(x+w, y));
      points.push_back(Point(x, y+w));
      points.push_back(Point(x+w, y+w));
    }
    printf("%d\n", diameter2(points));
  }
  return 0;
}
```

平面区域

```
// LA3218/UVa1340 Find the Border
// Rujia Liu
// 注意：本题可以直接使用“卷包裹”法求出外轮廓。本程序只是为了演示PSLG的实现
#include<cstdio>
#include<vector>
#include<cmath>
#include<algorithm>
#include<cstring>
#include<cassert>
using namespace std;

const double eps = 1e-8;
double dcmp(double x) {
  if(fabs(x) < eps) return 0; else return x < 0 ? -1 : 1;
}

struct Point {
  double x, y;
  Point(double x=0, double y=0):x(x),y(y) { }
};

typedef Point Vector;

Vector operator + (Vector A, Vector B) {
  return Vector(A.x+B.x, A.y+B.y);
}

Vector operator - (Point A, Point B) {
  return Vector(A.x-B.x, A.y-B.y);
}
```

```
Vector operator * (Vector A, double p) {
  return Vector(A.x*p, A.y*p);
}

// 理论上这个“小于”运算符是错的，因为可能有三个点a，b，c，a和b很接近（即a<b好b<a都不成立），b和c很接近，但a和c不接近
// 所以使用这种“小于”运算符的前提是能排除上述情况
bool operator < (const Point& a, const Point& b) {
  return dcmp(a.x - b.x) < 0 || (dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) < 0);
}

bool operator == (const Point& a, const Point &b) {
  return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0;
}

double Dot(Vector A, Vector B) { return A.x*B.x + A.y*B.y; }
double Cross(Vector A, Vector B) { return A.x*B.y - A.y*B.x; }
double Length(Vector A) { return sqrt(Dot(A, A)); }

typedef vector<Point> Polygon;

Point GetLineIntersection(const Point& P, const Vector& v, const Point& Q, const Vector& w) {
  Vector u = P-Q;
  double t = Cross(w, u) / Cross(v, w);
  return P+v*t;
}

bool SegmentProperIntersection(const Point& a1, const Point& a2, const Point& b1, const Point& b2) {
  double c1 = Cross(a2-a1,b1-a1), c2 = Cross(a2-a1,b2-a1),
  c3 = Cross(b2-b1,a1-b1), c4=Cross(b2-b1,a2-b1);
  return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
}

bool OnSegment(Point p, Point a1, Point a2) {
  return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) < 0;
}

// 多边形的有向面积
double PolygonArea(Polygon poly) {
  double area = 0;
  int n = poly.size();
  for(int i = 1; i < n-1; i++)
    area += Cross(poly[i]-poly[0], poly[(i+1)%n]-poly[0]);
  return area/2;
}

struct Edge {
  int from, to; // 起点，终点，左边的面编号
  double ang;
};

const int maxn = 10000 + 10; // 最大边数

// 平面直线图（PSGL）实现
struct PSLG {
  int n, m, face_cnt;
  double x[maxn], y[maxn];
  vector<Edge> edges;
  vector<int> G[maxn];
  int vis[maxn*2];  // 每条边是否已经访问过
  int left[maxn*2]; // 左面的编号
  int prev[maxn*2]; // 相同起点的上一条边（即顺时针旋转碰到的下一条边）的编号

  vector<Polygon> faces;
  double area[maxn]; // 每个polygon的面积

  void init(int n) {
    this->n = n;
```

```
    for(int i = 0; i < n; i++) G[i].clear();
    edges.clear();
    faces.clear();
  }

  // 有向线段from->to的极角
  double getAngle(int from, int to) {
    return atan2(y[to]-y[from], x[to]-x[from]);
  }

  void AddEdge(int from, int to) {
    edges.push_back((Edge){from, to, getAngle(from, to)});
    edges.push_back((Edge){to, from, getAngle(to, from)});
    m = edges.size();
    G[from].push_back(m-2);
    G[to].push_back(m-1);
  }

  // 找出faces并计算面积
  void Build() {
    for(int u = 0; u < n; u++) {
      // 给从u出发的各条边按极角排序
      int d = G[u].size();
      for(int i = 0; i < d; i++)
        for(int j = i+1; j < d; j++) // 这里偷个懒，假设从每个点出发的线段不会太多
          if(edges[G[u][i]].ang > edges[G[u][j]].ang) swap(G[u][i], G[u][j]);
      for(int i = 0; i < d; i++)
        prev[G[u][(i+1)%d]] = G[u][i];
    }

    memset(vis, 0, sizeof(vis));
    face_cnt = 0;
    for(int u = 0; u < n; u++)
      for(int i = 0; i < G[u].size(); i++) {
        int e = G[u][i];
        if(!vis[e]) { // 逆时针找圈
          face_cnt++;
          Polygon poly;
          for(;;) {
            vis[e] = 1; left[e] = face_cnt;
            int from = edges[e].from;
            poly.push_back(Point(x[from], y[from]));
            e = prev[e^1];
            if(e == G[u][i]) break;
            assert(vis[e] == 0);
          }
          faces.push_back(poly);
        }
      }

    for(int i = 0; i < faces.size(); i++) {
      area[i] = PolygonArea(faces[i]);
    }
  }
};

PSLG g;

const int maxp = 100 + 5;
int n, c;
Point P[maxp];

Point V[maxp*(maxp-1)/2+maxp];

// 在V数组里找到点p
int ID(Point p) {
  return lower_bound(V, V+c, p) - V;
```

```
}

// 假定poly没有相邻点重合的情况，只需要删除三点共线的情况
Polygon simplify(const Polygon& poly) {
  Polygon ans;
  int n = poly.size();
  for(int i = 0; i < n; i++) {
    Point a = poly[i];
    Point b = poly[(i+1)%n];
    Point c = poly[(i+2)%n];
    if(dcmp(Cross(a-b, c-b)) != 0) ans.push_back(b);
  }
  return ans;
}

void build_graph() {
  c = n;
  for(int i = 0; i < n; i++)
    V[i] = P[i];

  vector<double> dist[maxp]; // dist[i][j]是第i条线段上的第j个点离起点（P[i]）的距离
  for(int i = 0; i < n; i++)
    for(int j = i+1; j < n; j++)
      if(SegmentProperIntersection(P[i], P[(i+1)%n], P[j], P[(j+1)%n])) {
        Point p = GetLineIntersection(P[i], P[(i+1)%n]-P[i], P[j], P[(j+1)%n]-P[j]);
        V[c++] = p;
        dist[i].push_back(Length(p - P[i]));
        dist[j].push_back(Length(p - P[j]));
      }

  // 为了保证“很接近的点”被看作同一个，这里使用了sort+unique的方法
  // 必须使用前面提到的“理论上是错误”的小于运算符，否则不能保证“很接近的点”在排序后连续排列
  // 另一个常见的处理方式是把坐标扩大很多倍（比如100000倍），然后四舍五入变成整点（计算完毕后再还原），用少许的精度损失换来鲁棒性和速度。
  sort(V, V+c);
  c = unique(V, V+c) - V;

  g.init(c); // c是平面图的点数
  for(int i = 0; i < c; i++) {
    g.x[i] = V[i].x;
    g.y[i] = V[i].y;
  }
  for(int i = 0; i < n; i++) {
    Vector v = P[(i+1)%n] - P[i];
    double len = Length(v);
    dist[i].push_back(0);
    dist[i].push_back(len);
    sort(dist[i].begin(), dist[i].end());
    int sz = dist[i].size();
    for(int j = 1; j < sz; j++) {
      Point a = P[i] + v * (dist[i][j-1] / len);
      Point b = P[i] + v * (dist[i][j] / len);
      if(a == b) continue;
      g.AddEdge(ID(a), ID(b));
    }
  }

  g.Build();

  Polygon poly;
  for(int i = 0; i < g.faces.size(); i++)
    if(g.area[i] < 0) { // 对于连通图，惟一一个面积小于零的面是无限面
      poly = g.faces[i];
      reverse(poly.begin(), poly.end()); // 对于内部区域来说，无限面多边形的各个顶点是顺时针的
      poly = simplify(poly); // 无限面多边形上可能会有相邻共线点
      break;
    }
```

```
  int m = poly.size();
  printf("%d\n", m);

  // 挑选坐标最小的点作为输出的起点
  int start = 0;
  for(int i = 0; i < m; i++)
    if(poly[i] < poly[start]) start = i;
  for(int i = start; i < m; i++)
    printf("%.4lf %.4lf\n", poly[i].x, poly[i].y);
  for(int i = 0; i < start; i++)
    printf("%.4lf %.4lf\n", poly[i].x, poly[i].y);
}

int main() {
  while(scanf("%d", &n) == 1 && n) {
    for(int i = 0; i < n; i++) {
      int x, y;
      scanf("%d%d", &x, &y);
      P[i] = Point(x, y);
    }
    build_graph();
  }
  return 0;
}
```

虚树

```
#include <iostream>
#include <sstream>
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <vector>

const int N = 3e5 + 10;

int n;
std::vector<int> edge[N];
int father[N][21];
int size[N], deep[N];
int pos[N], tot;

void clear() {
    tot = 0;
    for (int i = 1; i <= n; i ++) {
        edge[i].clear();
    }
}

void init() {
    std::cin >> n;
    clear();
    for (int i = 1; i <= n - 1; i ++) {
        int u, v;
        scanf("%d%d", &u, &v);
        edge[u].push_back(v);
        edge[v].push_back(u);
    }
}

void dfs(int u) {
    size[u] = 1;
    pos[u] = ++tot;
```

```
    for (int i = 0; i < (int)edge[u].size(); i ++) {
        int v = edge[u][i];
        if (v == father[u][0]) {
            continue;
        }
        father[v][0] = u;
        deep[v] = deep[u] + 1;
        dfs(v);
        size[u] += size[v];
    }
}

void prepare() {
    for (int j = 1; j <= 20; j ++) {
        for (int i = 1; i <= n; i ++) {
            father[i][j] = father[father[i][j - 1]][j - 1];
        }
    }
}

int least_common_ancestor(int u, int v) {
    if (deep[u] < deep[v]) {
        std::swap(u, v);
    }
    for (int i = 20; i >= 0; i --) {
        if (deep[father[u][i]] >= deep[v]) {
            u = father[u][i];
        }
    }
    if (u == v) {
        return u;
    }
    for (int i = 20; i >= 0; i --) {
        if (father[u][i] != father[v][i]) {
            u = father[u][i];
            v = father[v][i];
        }
    }
    return father[u][0];
}

bool cmp_pos(int x, int y) {
    return pos[x] < pos[y];
}

int get_ancestor(int u, int cnt) {
    for (int i = 20; i >= 0; i --) {
        if ((1 << i) <= cnt) {
            u = father[u][i];
            cnt -= (1 << i);
        }
    }
    return u;
}

int dist(int u, int v) {
    int t = least_common_ancestor(u, v);
    return deep[u] + deep[v] - 2 * deep[t];
}

void solve(std::vector<int> &query) {
    static int stack[N], fa[N];
    static std::vector<int> all;
    static std::pair<int, int> best[N];
    static int extra[N], ans[N];
    std::vector<int> rem = query;
```

```
int top = 0;
all.clear();

sort(query.begin(), query.end(), cmp_pos);

for (int i = 0; i < (int)query.size(); i ++) {
    int u = query[i];
    if (top == 0) {
        stack[++top] = u;
        all.push_back(u);
        best[u] = std::make_pair(0, u);
    } else {
        int lca = least_common_ancestor(u, stack[top]);
        for ( ; deep[stack[top]] > deep[lca]; top --) {
            if (deep[stack[top - 1]] <= deep[lca]) {
                fa[stack[top]] = lca;
            }
        }
        if (stack[top] != lca) {
            fa[lca] = stack[top];
            stack[++top] = lca;
            best[lca] = std::make_pair(n + 10, -1);
            all.push_back(lca);
        }
        fa[u] = stack[top];
        stack[++top] = u;
        all.push_back(u);
        best[u] = std::make_pair(0, u);
    }
}

sort(all.begin(), all.end(), cmp_pos);

static int length[N];
for (int i = 0; i < (int)all.size(); i ++) {
    int u = all[i];
    if (u != 1) {
        length[u] = deep[u] - deep[fa[u]];
    }
}

for (int i = (int)all.size() - 1; i > 0; i --) {
    int u = all[i];
    std::pair<int, int> tmp = best[u];
    tmp.first += length[u];
    best[fa[u]] = std::min(best[fa[u]], tmp);
}
for (int i = 1; i < (int)all.size(); i ++) {
    int u = all[i];
    std::pair<int, int> tmp = best[fa[u]];
    tmp.first += length[u];
    best[u] = std::min(best[u], tmp);
}
/*
for (int i = 0; i < (int)all.size(); i ++) {
    printf("best[%d] = {%d, %d}\n", all[i], best[all[i]].first, best[all[i]].second);
    printf("size[%d] = %d\n", all[i], size[all[i]]);
}
*/
for (int i = 0; i < (int)query.size(); i ++) {
    ans[query[i]] = 0;
}

for (int i = 0; i < (int)all.size(); i ++) {
    int u = all[i];
    if (i == 0) {
        ans[best[u].second] = n - size[u];
```

```
        } else {
            int t = get_ancestor(u, length[u] - 1);
            if (best[u].second == best[fa[u]].second) {
                ans[best[u].second] += size[t] - size[u];
            } else {
                int step = u;
                for (int i = 20; i >= 0; i --) {
                    int mid = father[step][i];
                    if (deep[mid] <= deep[fa[u]]) {
                        continue;
                    }
                    std::pair<int, int> tmp1 = std::make_pair(dist(mid, best[u].second), best[u].second);
                    std::pair<int, int> tmp2 = std::make_pair(dist(mid, best[fa[u]].second), best[fa[u]].second);
                    if (tmp1 < tmp2) {
                        step = father[step][i];
                    }
                }
                ans[best[u].second] += size[step] - size[u];
                ans[best[fa[u]].second] += size[t] - size[step];
            }
        }
    }
    for (int i = 0; i < (int)all.size(); i ++) {
        int u = all[i];
        extra[u] = size[u];
    }
    for (int i = 0; i < (int)all.size(); i ++) {
        int u = all[i];
        int t = get_ancestor(u, length[u] - 1);
        extra[fa[u]] -= size[t];
    }
    for (int i = 0; i < (int)all.size(); i ++) {
        int u = all[i];
        ans[best[u].second] += extra[u];
    }

    for (int i = 0; i < (int)rem.size(); i ++) {
        printf("%d ", ans[rem[i]]);
    }
    printf("\n");
}

void work() {
    deep[1] = 1;
    dfs(1);
    prepare();

    int q;
    std::cin >> q;
    while (q --) {
        int cnt;
        static std::vector<int> cur;

        cur.clear();

        scanf("%d", &cnt);
        for (int i = 1; i <= cnt; i ++) {
            int t;
            scanf("%d", &t);
            cur.push_back(t);
        }

        solve(cur);
    }
}

int main() {
```

```
    //freopen("input.txt", "r", stdin);

    init();
    work();

    return 0;
}
```

动态树

```
#include <iostream>
#include <sstream>
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <vector>

const int N = 3e5 + 10;

int n;

struct LinkCutTree {
    struct Node {
        int value, max, inc;
        bool rev;
        int father, child[2];

        Node() {
        }
    };
    Node node[N];
    const Node EMPTY;

    void clear() {
        std::fill(node + 1, node + n + 1, EMPTY);
    }

    void __inc(int x, int delta) {
        if (x == 0) {
            return ;
        }
        node[x].inc += delta;
        node[x].value += delta;
        node[x].max += delta;
    }

    void update(int x) {
        if (x == 0) {
            return ;
        }
        if (node[x].inc != 0) {
            __inc(node[x].child[0], node[x].inc);
            __inc(node[x].child[1], node[x].inc);
            node[x].inc = 0;
        }
        if (node[x].rev == true) {
            std::swap(node[x].child[0], node[x].child[1]);
            node[node[x].child[0]].rev ^= true;
            node[node[x].child[1]].rev ^= true;
            node[x].rev = false;
        }
    }
```

```
void renew(int x) {
    update(x);
    update(node[x].child[0]);
    update(node[x].child[1]);
    node[x].max = std::max(node[x].value, std::max(node[node[x].child[0]].max, node[node[x].child[1]].max));
}

void change_value(int x, int value) {
    splay(x);
    node[x].value = node[x].max = value;
    renew(x);
}

bool is_splay_father(int y, int x) {
    return (y != 0) && (node[y].child[0] == x || node[y].child[1] == x);
}

void rotate(int x, int c) {
    int y = node[x].father;
    node[y].child[c ^ 1] = node[x].child[c];
    if (node[x].child[c] != 0) {
        node[node[x].child[c]].father = y;
    }
    node[x].father = node[y].father;
    if (node[node[y].father].child[0] == y) {
        node[node[x].father].child[0] = x;
    } else if (node[node[y].father].child[1] == y) {
        node[node[x].father].child[1] = x;
    }
    node[x].child[c] = y;
    node[y].father = x;
    renew(y);
}

void splay(int x) {
    if (x == 0) {
        return ;
    }
    update(x);
    while (is_splay_father(node[x].father, x)) {
        int y = node[x].father;
        int z = node[y].father;
        if (is_splay_father(z, y)) {
            update(z);
            update(y);
            update(x);
            int c = (y == node[z].child[0]);
            if (x == node[y].child[c]) {
                rotate(x, c ^ 1);
                rotate(x, c);
            } else {
                rotate(y, c);
                rotate(x, c);
            }
        } else {
            update(y);
            update(x);
            rotate(x, x == node[y].child[0]);
            break;
        }
    }
    renew(x);
}

int access(int x) {
    int y = 0;
    for ( ; x != 0; x = node[x].father) {
```

```
            splay(x);
            node[x].child[1] = y;
            renew(y = x);
        }
        return y;
    }

    int get_root(int x) {
        x = access(x);
        while (true) {
            update(x);
            if (node[x].child[0] == 0) {
                break;
            }
            x = node[x].child[0];
        }
        return x;

    }

    void make_root(int x) {
        node[access(x)].rev ^= true;
        splay(x);
    }

    void link(int x, int y) {
        make_root(x);
        node[x].father = y;
        access(x);
    }

    void cut(int x, int y) {
        make_root(x);
        access(y);
        splay(y);
        node[node[y].child[0]].father = 0;
        node[y].child[0] = 0;
        renew(y);
    }

    void modify(int x, int y, int delta) {
        make_root(x);
        access(y);
        splay(y);
        __inc(y, delta);
    }

    int get_max(int x, int y) {
        make_root(x);
        access(y);
        splay(y);
        return node[y].max;
    }
};
LinkCutTree lct;

void clear() {
    lct.clear();
}

void init() {
    for (int i = 1; i <= n - 1; i ++) {
        int u, v;
        scanf("%d%d", &u, &v);
        lct.link(u, v);
    }
    for (int i = 1; i <= n; i ++) {
```

```
        int t;
        scanf("%d", &t);
        lct.change_value(i, t);
    }
}

void work() {
    int q;
    std::cin >> q;

    while (q --) {
        int type;
        scanf("%d", &type);

        if (type == 1) {
            int u, v;
            scanf("%d%d", &u, &v);
            if (lct.get_root(u) == lct.get_root(v)) {
                puts("-1");
            } else {
                lct.link(u, v);
            }
        } else if (type == 2) {
            int u, v;
            scanf("%d%d", &u, &v);
            if (u == v || lct.get_root(u) != lct.get_root(v)) {
                puts("-1");
            } else {
                lct.cut(u, v);
            }
        } else if (type == 3) {
            int delta, u, v;
            scanf("%d%d%d", &delta, &u, &v);
            if (lct.get_root(u) != lct.get_root(v)) {
                puts("-1");
            } else {
                lct.modify(u, v, delta);
            }
        } else {
            int u, v;
            scanf("%d%d", &u, &v);
            if (lct.get_root(u) != lct.get_root(v)) {
                puts("-1");
            } else {
                printf("%d\n", lct.get_max(u, v));
            }
        }
    }
    printf("\n");
}

int main() {
    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);

    while (std::cin >> n) {
        clear();
        init();
        work();
    }

    return 0;
}
```

KM算法(N^3)

```
#include <iostream>
#include <sstream>
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <vector>

const int N = 25;
const int INF = 1e8;

int n;
int profit[2][N][N];
int answer = -INF;

struct KM_State {
    int lx[N], ly[N];
    int match[N], way[N];

    KM_State() {
        for (int i = 1; i <= n; i ++) {
            match[i] = 0;
            lx[i] = 0;
            ly[i] = 0;
            way[i] = 0;
        }
    }
};

struct KM_Solver {
    int w[N][N];
    KM_State state;
    int slack[N];
    bool used[N];

    KM_Solver() {
        for (int i = 1; i <= n; i ++) {
            for (int j = 1; j <= n; j ++) {
                w[i][j] = 0;
            }
        }
    }

    void hungary(int x) {
        state.match[0] = x;
        int j0 = 0;
        for (int j = 0; j <= n; j ++) {
            slack[j] = INF;
            used[j] = false;
        }
        do {
            used[j0] = true;
            int i0 = state.match[j0], delta = INF, j1;
            for (int j = 1; j <= n; j ++) {
                if (used[j] == false) {
                    int cur = w[i0][j] - state.lx[i0] - state.ly[j];
                    if (cur < slack[j]) {
                        slack[j] = cur;
                        state.way[j] = j0;
                    }
                    if (slack[j] < delta) {
                        delta = slack[j];
                        j1 = j;
                    }
                }
```

```
            }
            for (int j = 0; j <= n; j ++) {
                if (used[j]) {
                    state.lx[state.match[j]] += delta;
                    state.ly[j] -= delta;
                } else {
                    slack[j] -= delta;
                }
            }
            j0 = j1;
        } while (state.match[j0] != 0);

        do {
            int j1 = state.way[j0];
            state.match[j0] = state.match[j1];
            j0 = j1;
        } while (j0);
    }

    int get_ans() {
        int ret = 0;
        for (int i = 1; i <= n; i ++) {
            if (state.match[i] > 0) {
                ret += w[state.match[i]][i];
            }
        }
        return state.ly[0];
    }
};

void init() {
    std::cin >> n;
    for (int t = 0; t <= 1; t ++) {
        for (int i = 1; i <= n; i ++) {
            for (int j = 1; j <= n; j ++) {
                scanf("%d", &profit[t][i][j]);
            }
        }
    }
}

void dfs(int x, int y, KM_Solver &solver) {
    if (x + y == n) {
        answer = std::max(answer, solver.get_ans());
        return ;
    }
    if (2 * x + 2 <= n) {
        KM_State tmp = solver.state;
        for (int i = 1; i <= n; i ++) {
            solver.w[x + y + 1][i] = -profit[0][x + y + 1][i];
        }
        solver.hungary(x + y + 1);
        dfs(x + 1, y, solver);
        solver.state = tmp;
    }
    if (2 * y + 2 <= n) {
        KM_State tmp = solver.state;
        for (int i = 1; i <= n; i ++) {
            solver.w[x + y + 1][i] = -profit[1][x + y + 1][i];
        }
        solver.hungary(x + y + 1);
        dfs(x, y + 1, solver);
        solver.state = tmp;
    }
}

void work() {
```

```
    static KM_Solver solver;
    dfs(0, 0, solver);
    std::cout << answer << std::endl;
}

int main() {
    //freopen("C.in", "r", stdin);

    init();
    work();

    return 0;
}
```

最小树形图

```
// UVa11865 Stream My Contest
// Rujia Liu
#include<cstdio>
#include<cstring>
#include<vector>
#include<algorithm>
using namespace std;

const int INF = 1000000000;
const int maxn = 100 + 10;

// 固定根的最小树型图，邻接矩阵写法
struct MDST {
  int n;
  int w[maxn][maxn]; // 边权
  int vis[maxn];     // 访问标记，仅用来判断无解
  int ans;           // 计算答案
  int removed[maxn]; // 每个点是否被删除
  int cid[maxn];     // 所在圈编号
  int pre[maxn];     // 最小入边的起点
  int iw[maxn];      // 最小入边的权值
  int max_cid;       // 最大圈编号

  void init(int n) {
    this->n = n;
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++) w[i][j] = INF;
  }

  void AddEdge(int u, int v, int cost) {
    w[u][v] = min(w[u][v], cost); // 重边取权最小的
  }

  // 从s出发能到达多少个结点
  int dfs(int s) {
    vis[s] = 1;
    int ans = 1;
    for(int i = 0; i < n; i++)
      if(!vis[i] && w[s][i] < INF) ans += dfs(i);
    return ans;
  }

  // 从u出发沿着pre指针找圈
  bool cycle(int u) {
    max_cid++;
    int v = u;
    while(cid[v] != max_cid) { cid[v] = max_cid; v = pre[v]; }
    return v == u;
  }
```

```
  // 计算u的最小入弧，入弧起点不得在圈c中
  void update(int u) {
    iw[u] = INF;
    for(int i = 0; i < n; i++)
      if(!removed[i] && w[i][u] < iw[u]) {
        iw[u] = w[i][u];
        pre[u] = i;
      }
  }

  // 根结点为s，如果失败则返回false
  bool solve(int s) {
    memset(vis, 0, sizeof(vis));
    if(dfs(s) != n) return false;

    memset(removed, 0, sizeof(removed));
    memset(cid, 0, sizeof(cid));
    for(int u = 0; u < n; u++) update(u);
    pre[s] = s; iw[s] = 0; // 根结点特殊处理
    ans = max_cid = 0;
    for(;;) {
      bool have_cycle = false;
      for(int u = 0; u < n; u++) if(u != s && !removed[u] && cycle(u)){
        have_cycle = true;
        // 以下代码缩圈，圈上除了u之外的结点均删除
        int v = u;
        do {
          if(v != u) removed[v] = 1;
          ans += iw[v];
          // 对于圈外点i，把边i->v改成i->u（并调整权值）；v->i改为u->i
          // 注意圈上可能还有一个v'使得i->v'或者v'->i存在，因此只保留权值最小的i->u和u->i
          for(int i = 0; i < n; i++) if(cid[i] != cid[u] && !removed[i]) {
            if(w[i][v] < INF) w[i][u] = min(w[i][u], w[i][v]-iw[v]);
            w[u][i] = min(w[u][i], w[v][i]);
            if(pre[i] == v) pre[i] = u;
          }
          v = pre[v];
        } while(v != u);
        update(u);
        break;
      }
      if(!have_cycle) break;
    }
    for(int i = 0; i < n; i++)
      if(!removed[i]) ans += iw[i];
    return true;
  }
};

//////// 题目相关
MDST solver;

struct Edge {
  int u, v, b, c;
  bool operator < (const Edge& rhs) const {
    return b > rhs.b;
  }
};

const int maxm = 10000 + 10;
int n, m, C;
Edge edges[maxm];

// 取b前cnt大的边构造网络，判断最小树型图的边权和是否小于C
bool check(int cnt) {
  solver.init(n);
```

```
  for(int i = 0; i < cnt; i++)
    solver.AddEdge(edges[i].u, edges[i].v, edges[i].c);
  if(!solver.solve(0)) return false;
  return solver.ans <= C;
}

int main() {
  int T;
  scanf("%d", &T);
  while(T--) {
    scanf("%d%d%d", &n, &m, &C);
    for(int i = 0; i < m; i++) {
      scanf("%d%d%d%d", &edges[i].u, &edges[i].v, &edges[i].b, &edges[i].c);
    }
    sort(edges, edges+m);
    int L = 1, R = m, ans = -1;
    while(L <= R) {
      int M = L + (R-L)/2;
      if(check(M)) { ans = edges[M-1].b; R = M-1; }
      else L = M+1;
    }
    if(ans < 0) printf("streaming not possible.\n");
    else printf("%d kbps\n", ans);
  }
  return 0;
}
```

可持久化线段树

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <vector>
using namespace std;

const int N = 100005;

struct Node
{
    int best[3], son[2];
};

struct Segment_Tree
{
    Node tree[N * 19];
    int __root[N], tot;
    void merge(Node &ret, int left, int right, Node t1, Node t2)
    {
        ret.best[0] = max(t1.best[2] + t2.best[1], max(t1.best[0], t2.best[0]));
        int mid = (left + right) >> 1;
        ret.best[1] = t1.best[1] + (t1.best[1] == mid - left + 1) * t2.best[1];
        ret.best[2] = t2.best[2] + (t2.best[2] == right - mid) * t1.best[2];
    }
    void add(int &root, int root1, int left, int right, int x)
    {
        root = ++tot;
        if(left == x && right == x)
        {
            tree[root].best[0] = tree[root].best[1] = tree[root].best[2] = 1;
            return ;
        }
        int mid = (left + right) >> 1;
```

```
        if(x <= mid)
        {
            add(tree[root].son[0], tree[root1].son[0], left, mid, x);
            tree[root].son[1] = tree[root1].son[1];
        }
        else
        {
            tree[root].son[0] = tree[root1].son[0];
            add(tree[root].son[1], tree[root1].son[1], mid + 1, right, x);
        }
        merge(tree[root], left, right, tree[tree[root].son[0]], tree[tree[root].son[1]]);
    }
    Node search(int root, int left, int right, int L,int R)
    {
        if(root == 0) return tree[0];
        if(left == L && right == R) return tree[root];
        int mid = (left + right) >> 1;
        if(R <= mid) return search(tree[root].son[0], left, mid, L, R);
        if(L > mid) return search(tree[root].son[1], mid + 1, right, L, R);
        Node t1 = search(tree[root].son[0], left, mid, L, mid);
        Node t2 = search(tree[root].son[1], mid + 1, right, mid + 1, R);
        Node ret;
        merge(ret, left, right, t1, t2);
        return ret;
    }
};
Segment_Tree T;

int n, q;
pair<int, int> g[N];

void insert(int x, int pos)
{
    T.add(T.__root[x], T.__root[x - 1], 1, n, pos);
}
void init()
{
    cin >> n;
    for(int i = 1; i <= n; i ++)
        scanf("%d", &g[i].first), g[i].first *= -1, g[i].second = i;
}
bool check(int mid, int L, int R, int length)
{
    Node step = T.search(T.__root[mid], 1, n, L, R);
    return (step.best[0] >= length);
}
int Solve(int L, int R, int length)
{
    int low = 1, high = n, ret = n + 1;
    while(low <= high)
    {
        int mid = (low + high) >> 1;
        if(check(mid, L, R, length)) ret = min(ret, mid), high = mid - 1;
        else low = mid + 1;
    }
    return ret;
}
void work()
{
    sort(g + 1, g + n + 1);
    for(int i = 1; i <= n; i ++)
        insert(i, g[i].second);
    cin >> q;
    while(q --)
    {
        int c, d, e;
        scanf("%d%d%d", &c, &d, &e);
```

```
        printf("%d\n", -g[Solve(c, d, e)].first);
    }
}

int main()
{
    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);
    init(), work();
    return 0;
}
```

Hint of pb_ds

priority_queue:

```
#include <ext/pb_ds/priority_queue.hpp>

__gnu_pbds::priority_queue<int> heap;

* point iterator push(const reference)
* void modify(point iterator, const reference)
* void erase(point iterator)

* void join(priority queue &other)
* 注意：other会被清空
```

tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

tree<int, int, less<int>, rb_tree_tag, tree_order_statistics_node_update>

* find_by_order(size type order)
* size_type order of key(const key reference r key)

void join(tree &other)
void split(const key reference r key, tree &other)
```