****1 图论****

**1.1基本定理**

**1.1.1平面图欧拉定理**

在一个由若干顶点和它们之间的一些不相交的边所组成的图中，等式V+F=E+2总成立，其中V表示顶点个数，E表示总的边数，F表示这个图分割出来的区域个数（包括一个“外部区域”，例如一个圆把平面分割为两个区域）。

**1.1.2二分图相关**

**最大匹配=最小顶点覆盖**

**最大独立集=顶点个数-最小顶点覆盖**

**最小路径覆盖=二分图最大独立集**

**二分图中的桥？poj\_1904,hdu\_4685**

**构建完备匹配->建反向边->求强连通分量**

**1.1.3生成树相关**

**生成树定理**

**G的度数矩阵D[G]是一个n\*n的矩阵，并且满足：当i≠j时,dij=0；当i=j时，dij等于vi的度数。**

**G的邻接矩阵A[G]是一个n\*n的矩阵，并且满足：如果vi、vj之间有边直接相连，则aij=1，否则为0。**

**我们定义G的Kirchhoff矩阵(也称为拉普拉斯算子)C[G]为C[G]=D[G]-A[G]，则Matrix-Tree定理可以描述为：G的所有不同的生成树的个数等于其Kirchhoff矩阵C[G]任何一个n-1阶主子式的行列式的绝对值。所谓n-1阶主子式，就是对于r(1≤r≤n)，将C[G]的第r行、第r列同时去掉后得到的新矩阵，用Cr[G]表示。**

**次小生成树？hdu\_4081**

**K度最小生成树？poj\_1639**

**1.1.4最大流与最小割相关**

**SRM570 900**

**1.1.5其他**

|  |
| --- |
| **/\*初始化\*/**  **#define Maxm 200010**  **#define Maxn 100010**  **int head[Maxn], cnt;**  **struct edge {**  **int u, v, w, next;**  **} e[Maxm];**  **void init() {**  **memset(head, -1, sizeof(head));**  **cnt = 0;**  **}**  **void addedge(int u, int v) {**  **e[cnt].u = u, e[cnt].v = v, e[cnt].next = head[u], head[u] = cnt ++ ;**  **e[cnt].u = v, e[cnt].v = u, e[cnt].next = head[v], head[v] = cnt ++ ;**  **}** |

**1.2 拓扑排序**

|  |
| --- |
| **/\* 拓扑排序 O(e)**  **\* 确保是有向无环图！**  **\* 结果逆序存放在sta中！**  **\* \*/**  **VI ve[Maxn];**  **bool vis[Maxn];**  **int sta[Maxn], top;**  **void dfsTopo(int u) {**  **vis[u] = true;**  **for (int i = 0; i < ve[u].size(); i ++ )**  **if (!vis[ve[u][i]])**  **dfsTopo(ve[u][i]);**  **sta[top ++ ] = u;**  **}**  **void topoSort(int n) {**  **memset(vis, 0, sizeof(vis));**  **top = 0;**  **for (int i = 1; i <= n; i ++ )**  **if (!vis[i]) dfsTopo(i);**  **}** |

**1.3 最短路**

|  |
| --- |
| **/\* 最短路**  **\* dijkstra算法 O(n ^ 2)**  **\* SPFA算法 O(e \* K)**  **\* \*/**  **int dis[Maxn];**  **void spfa(int n, int s) {**  **priority\_queue <PII> que;**  **memset(dis, 0x3f, sizeof(dis));**  **dis[s] = 0;**  **que.push(MP(- dis[s], s));**  **while (!que.empty()) {**  **int u = que.top().B, d = - que.top().A;**  **que.pop();**  **if (d != dis[u]) continue;**  **for (int i = head[u]; i != -1; i = e[i].next)**  **if (d + e[i].w < dis[e[i].v]) {**  **dis[e[i].v] = d + e[i].w;**  **que.push(MP( - dis[e[i].v], e[i].v));**  **}**  **}**  **}**  **bool vis[Maxn];**  **bool dfsSpfa(int u) { // 可用于判负权环，注意初始化 dis 和 vis 数组**  **vis[u] = true;**  **for (int i = head[u]; i != -1; i = e[i].next)**  **if (dis[u] + e[i].w < dis[e[i].v]) {**  **dis[e[i].v] = dis[u] + e[i].w;**  **if (vis[e[i].v]) return true;**  **if (dfsSpfa(e[i].v)) return true;**  **}**  **vis[u] = false;**  **return false;**  **}** |

**1.4 强连通分量**

|  |
| --- |
| **/\* 强连通分量**  **\* Tarjan算法 O(e)**  **\* 有向图缩点, 缩成一个有向无环图**  **\* \*/**  **int dfn[Maxn], low[Maxn], block[Maxn], sta[Maxn], top, tsp, N, insta[Maxn];**  **VI ne[Maxn];**  **/\*初始化 top = tsp = N = 0 和 block为 0, dfn 为 0\*/**  **void tarjan(int u) {**  **dfn[u] = low[u] = ++ tsp;**  **sta[top ++ ] = u;**  **insta[u] = 1;**  **for (int i = head[u]; i != -1; i = e[i].next) {**  **int v = e[i].v;**  **if (!dfn[v]) {**  **tarjan(v);**  **low[u] = min(low[u], low[v]);**  **}**  **else {**  **if (!insta[v]) continue;**  **low[u] = min(low[u], dfn[v]);**  **}**  **}**  **if (low[u] == dfn[u]) {**  **N ++ ;**  **int v;**  **do {**  **v = sta[ -- top];**  **insta[v] = 0;**  **block[v] = N;**  **} while (v != u);**  **}**  **}**  **void buildNewGraph(int n) {**  **memset(dfn, 0, sizeof(dfn));**  **memset(block, 0, sizeof(block));**  **memset(insta, 0, sizeof(insta));**  **top = N = tsp = 0;**  **for (int i = 1; i <= n; i ++ )**  **if (!dfn[i])**  **tarjan(i);**  **for (int i = 1; i <= N; i ++ )**  **ne[i].clear();**  **for (int i = 0; i < cnt; i ++ )**  **if (block[e[i].u] != block[e[i].v])**  **ne[block[e[i].u]].PB(block[e[i].v]);**  **for (int i = 1; i <= N; i ++ )**  **ne[i].erase(unique(ne[i].begin(), ne[i].end()), ne[i].end());**  **}** |

**1.5 2-sat**

|  |
| --- |
| **/\*2-sat\*/**  **vector<VI> e, ne;**  **int dfn[Maxn], low[Maxn], block[Maxn], sta[Maxn], top, tsp, N;**  **int sta1[Maxn], rid[Maxn], top1, insta[Maxn];**  **//rid 新旧图映射，sta1, top1 拓扑排序，输出路径**  **/\*初始化 top = tsp = N = 0 和 block为 -1, dfn 为 0\*/**  **void tarjan(int u) {**  **dfn[u] = low[u] = ++ tsp;**  **sta[top ++ ] = u;**  **insta[u] = 1;**  **for (int i = 0; i < e[u].size(); i ++ ) {**  **int v = e[u][i];**  **if (!dfn[v]) {**  **tarjan(v);**  **low[u] = min(low[u], low[v]);**  **}**  **else {**  **if (!insta[v]) continue;**  **low[u] = min(low[u], dfn[v]);**  **}**  **}**  **if (low[u] == dfn[u]) {**  **N ++ ;**  **int v;**  **do {**  **v = sta[ -- top];**  **insta[v] = 0;**  **block[v] = N;**  **} while (v != u);**  **rid[N] = u;**  **sta1[top1 ++ ] = N;**  **N ++ ;**  **}**  **}**  **int color[Maxn];**  **void colorBlue(int u) {**  **color[u] = 2;**  **for (int i = 0; i < ne[u].size(); i ++ ) {**  **int v = ne[u][i];**  **if (!color[v])**  **colorBlue(v);**  **}**  **}**  **bool twoSat(int n) {**  **memset(insta, 0, sizeof(insta));**  **memset(dfn, 0, sizeof(dfn));**  **memset(block, -1, sizeof(block));**  **top1 = top = N = tsp = 0;**  **for (int i = 0; i < n \* 2; i ++ )**  **if (!dfn[i])**  **tarjan(i);**  **for (int i = 0; i < n \* 2; i += 2 )**  **if (block[i] == block[i ^ 1])**  **return 0;**  **//return 1;**  **/\*建立新图\*/**  **ne.clear();**  **for (int i = 0; i < N; i ++ ) {**  **VI p;**  **ne.PB(p);**  **}**  **for (int i = 0; i < n \* 2; i ++ )**  **for (int j = 0; j < e[i].size(); j ++ )**  **if (block[i] != block[e[i][j]])**  **ne[block[e[i][j]]].PB(block[i]); //反向建图**  **memset(color, 0, sizeof(color));**  **for (int i = 0; i < top1; i ++ ) {**  **int x = sta1[i];**  **if (!color[x]) {**  **color[x] = 1;**  **x = block[rid[x] ^ 1];**  **colorBlue(x);**  **}**  **}**  **return 1;**  **}** |

**1.6双连通分量与LCA**

**1.6.1点双连通分量**

|  |
| --- |
| **/\*点双连通分量、割点、倍增LCA**  **孤立点需特判\*/**  **#define Maxm 200010**  **#define Maxn 10010**  **int head[Maxn], cnt;**  **struct edge {**  **int u, v, w, next;**  **} e[Maxm];**  **void init() {**  **memset(head, -1, sizeof(head));**  **cnt = 0;**  **}**  **void addedge(int u, int v) {**  **e[cnt].u = u, e[cnt].v = v, e[cnt].next = head[u], head[u] = cnt ++ ;**  **e[cnt].u = v, e[cnt].v = u, e[cnt].next = head[v], head[v] = cnt ++ ;**  **}**  **int n, m, block[Maxm];**  **int tsp, dfn[Maxn], low[Maxn], n1, iscut[Maxn];**  **stack<int> sta;**  **void tarjan(int u, int pre) {**  **int v, child = 0;**  **dfn[u] = low[u] = ++ tsp;**  **for (int i = head[u]; i != -1; i = e[i].next) {**  **if (i == pre) continue;**  **v = e[i].v;**  **if (dfn[v] < dfn[u]) {**  **sta.push(i);**  **if (!dfn[v]) {**  **child ++ ;**  **tarjan(v, i ^ 1);**  **low[u] = min(low[u], low[v]);**  **if (low[v] >= dfn[u]) { //割点判断**  **while (sta.top() != i) {**  **block[sta.top() / 2] = n1;**  **sta.pop();**  **}**  **block[i / 2] = n1 ++ ;**  **sta.pop();**  **iscut[u] = 1;**  **}**  **}**  **else low[u] = min(low[u], low[v]);**  **}**  **}**  **if (pre < 0 && child == 1) iscut[u] = -1;**  **}**  **VI ne[Maxn \* 3]; //新图**  **bool color[Maxn \* 3];**  **void solve() {**  **memset(dfn, 0, sizeof(dfn));**  **memset(low, 0, sizeof(low));**  **memset(iscut, -1, sizeof(iscut));**  **memset(color, 0, sizeof(color));**  **n1 = 0;**  **for (int i = 1; i <= n; i ++ )**  **if (!dfn[i]) {**  **tsp = 0;**  **while (!sta.empty()) sta.pop();**  **tarjan(i, -1);**  **}**  **/\*建立新图\*/**  **for (int i = 1; i <= n; i ++ )**  **if (iscut[i] == 1) {**  **color[n1] = true;**  **iscut[i] = n1 ++ ;**  **}**  **for (int i = 0; i < n1; i ++ )**  **ne[i].clear();**  **for (int i = 0; i < cnt / 2; i ++ )**  **if (iscut[e[i \* 2].u] == -1 && iscut[e[i \* 2].v] == -1) continue;**  **else {**  **int u, v;**  **if (iscut[e[i \* 2].u] != -1) {**  **u = iscut[e[i \* 2].u];**  **v = block[i];**  **ne[u].PB(v);**  **ne[v].PB(u);**  **}**  **if (iscut[e[i \* 2].v] != -1){**  **u = iscut[e[i \* 2].v];**  **v = block[i];**  **ne[u].PB(v);**  **ne[v].PB(u);**  **}**  **}**  **}** |

**1.6.2 LCA**

|  |
| --- |
| **/\*LCA\*/**  **int fa[Maxn \* 3], sum[Maxn \* 3], h[Maxn \* 3];**  **int anc[Maxn \* 3][20];**  **bool vis[Maxn \* 3];**  **void dfs(int t) {**  **vis[t] = true;**  **if (color[t])**  **sum[t] ++ ;**  **for (int i = 0; i < ne[t].size(); i ++ )**  **if (!vis[ne[t][i]]) {**  **fa[ne[t][i]] = t;**  **sum[ne[t][i]] = sum[t];**  **h[ne[t][i]] = h[t] + 1;**  **dfs(ne[t][i]);**  **}**  **}**  **void lca\_prepare() {**  **memset(vis, 0, sizeof(vis));**  **memset(sum, 0, sizeof(sum));**  **memset(h, 0, sizeof(h));**  **for (int i = 0; i < n1; i ++ )**  **if (!vis[i]) {**  **fa[i] = -1;**  **dfs(i);**  **}**  **memset(anc, -1, sizeof(anc));**  **for (int i = 0; i < n1; i ++ )**  **anc[i][0] = fa[i];**  **for (int j = 1; ; j ++ ) {**  **bool flag = false;**  **for (int i = 0; i < n1; i ++ ) {**  **if (anc[i][j - 1] != -1)**  **anc[i][j] = anc[anc[i][j - 1]][j - 1];**  **if (anc[i][j] != -1)**  **flag = true;**  **}**  **if (!flag) break;**  **}**  **}**  **int query\_lca(int u, int v) {**  **int dif = abs(h[u] - h[v]), i = 0;**  **if (h[u] < h[v]) swap(u, v);**  **while (dif) {**  **if (dif & 1) u = anc[u][i];**  **i ++ ;**  **dif >>= 1;**  **}**  **for (i = 19; i >= 0; i -- )**  **if (anc[u][i] == anc[v][i]) continue;**  **else u = anc[u][i], v = anc[v][i];**  **if (u == v) return u;**  **else return anc[u][0];**  **}** |

**1.6.3边双连通分量**

|  |
| --- |
| **/\*边双连通分量、桥\*/**  **#define Maxm 2000010**  **#define Maxn 200010**  **int head[Maxn], cnt;**  **struct edge {**  **int u, v, next;**  **} e[Maxm];**  **void init() {**  **memset(head, -1, sizeof(head));**  **cnt = 0;**  **}**  **void addedge(int u, int v) {**  **e[cnt].u = u, e[cnt].v = v, e[cnt].next = head[u], head[u] = cnt ++ ;**  **e[cnt].u = v, e[cnt].v = u, e[cnt].next = head[v], head[v] = cnt ++ ;**  **}**  **int tsp, dfn[Maxn], low[Maxn], N, isbr[Maxm], block[Maxn];**  **stack<int> sta;**  **void tarjan(int u, int pre) {**  **int v, child = 0;**  **dfn[u] = low[u] = ++ tsp;**  **for (int i = head[u]; i != -1; i = e[i].next) {**  **if (i == pre) continue;**  **v = e[i].v;**  **if (dfn[v] < dfn[u]) {**  **if (!dfn[v]) {**  **sta.push(v);**  **child ++ ;**  **tarjan(v, i ^ 1);**  **low[u] = min(low[u], low[v]);**  **if (low[v] > dfn[u]) { //桥的判断**  **while (sta.top() != v) {**  **block[sta.top()] = N;**  **sta.pop();**  **}**  **block[v] = N ++ ;**  **sta.pop();**  **isbr[i] = true;**  **}**  **}**  **else low[u] = min(low[u], low[v]);**  **}**  **}**  **}**  **VI ne[Maxn \* 3];**  **void solve(int n) {**  **memset(dfn, 0, sizeof(dfn));**  **memset(low, 0, sizeof(low));**  **memset(isbr, 0, sizeof(isbr));**  **memset(block, -1, sizeof(block));**  **N = 0;**  **for (int i = 1; i <= n; i ++ )**  **if (!dfn[i]) {**  **tsp = 0;**  **while (!sta.empty()) sta.pop();**  **sta.push(i);**  **tarjan(i, -1);**  **while (!sta.empty()) {**  **block[sta.top()] = N;**  **sta.pop();**  **}**  **N ++ ;**  **}**  **//建立新图**  **for (int i = 0; i < N; i ++ )**  **ne[i].clear();**  **for (int i = 0; i < cnt; i ++ )**  **if (isbr[i]) {**  **int u = block[e[i].u];**  **int v = block[e[i].v];**  **ne[u].PB(v);**  **ne[v].PB(u);**  **}**  **}** |

**1.7最小树形图**

|  |
| --- |
| **/\* 最小树形图（根固定） O(VE)**  **\* 有向图最小生成树**  **\* 根不固定，添加一个根节点与所有点连无穷大的边！**  **\* 根据pre的信息能构造出这棵树！**  **\* \*/**  **int vis[Maxn], pre[Maxn], belong[Maxn], in[Maxn];**  **int dirtree(int n, int root) {**  **int sum = 0;**  **while (1) {**  **for (int i = 0; i < n; i ++ ) {**  **in[i] = inf;**  **pre[i] = -1;**  **belong[i] = -1;**  **vis[i] = -1;**  **}**  **in[root] = 0;**  **for (int i = 0; i < cnt; i ++ ) //除原点外，找每个点的最小入边**  **if (e[i].u != e[i].v) {**  **if (e[i].w < in[e[i].v]) {**  **in[e[i].v] = e[i].w;**  **pre[e[i].v] = e[i].u;**  **}**  **}**  **for (int i = 0; i < n; i ++ )**  **if (in[i] == inf) return -1;**  **int num = 0;**  **for (int i = 0; i < n; i ++ ) { //找圈，收缩圈**  **if (vis[i] == -1) {**  **int j = i;**  **while (j != -1 && vis[j] == -1 && j != root) {**  **vis[j] = i;**  **j = pre[j];**  **}**  **if (j != -1 && vis[j] == i) {**  **for (int k = pre[j]; k != j; k = pre[k])**  **belong[k] = num;**  **belong[j] = num ++ ;**  **}**  **}**  **sum += in[i];**  **}**  **if (num == 0) return sum;**  **for (int i = 0; i < n; i ++ )**  **if (belong[i] == -1)**  **belong[i] = num ++ ;**  **for (int i = 0; i < cnt; i ++ ) { //重新构图**  **e[i].w = e[i].w - in[e[i].v];**  **e[i].v = belong[e[i].v];**  **e[i].u = belong[e[i].u];**  **}**  **n = num;**  **root = belong[root];**  **}**  **}** |

**1.8最大流算法**

**1.8.1 EK**

|  |
| --- |
| **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EK算法\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **int c[Maxn][Maxn], pre[Maxn], que[Maxn][2], n, m, s, t;**  **void Init() {**  **memset(c, 0, sizeof(c));**  **scanf("%d%d", &n, &m);**  **for (int i = 0; i < m; i ++ ) {**  **int u, v, w;**  **scanf("%d%d%d", &u, &v, &w);**  **c[u][v] += w;**  **//c[v][u] += w;**  **}**  **}**  **bool vis[Maxn];**  **bool Extend(int s, int t, int &r) {**  **int idx = s, l;**  **l = r = 0;**  **memset(vis, 0, sizeof(vis));**  **que[r][1] = INF;**  **que[r ++ ][0] = idx;**  **pre[idx] = idx;**  **vis[idx] = true;**  **while (l < r) {**  **idx = que[l][0];**  **for (int i = 1; i <= n; i ++ )**  **if (!vis[i] && c[idx][i] > 0) {**  **que[r][1] = min(que[l][1], c[idx][i]);**  **que[r][0] = i;**  **pre[i] = idx;**  **if (i == t) return true;**  **r ++ ;**  **vis[i] = true;**  **}**  **l ++ ;**  **}**  **return false;**  **}**  **void Fill(int s, int t, int w) {**  **int u, v = t;**  **while (v != s) {**  **u = pre[v];**  **c[u][v] -= w;**  **c[v][u] += w;**  **v = u;**  **}**  **}**  **int EK(int s, int t) {**  **int tail = 0, flow = 0;**  **while (Extend(s, t, tail)) {**  **flow += que[tail][1];**  **Fill(s, t, que[tail][1]);**  **tail = 0;**  **}**  **return flow;**  **}** |

**1.8.2 Dinic**

|  |
| --- |
| **/\*Dinic\*/**  **int dis[Maxn], cur[Maxn], sta[Maxn], que[Maxn], pre[Maxn];**  **bool bfs(int s, int t, int n) {**  **int front = 0, tail = 0;**  **memset(dis, -1, sizeof(dis[0]) \* (n + 1));**  **dis[s] = 0;**  **que[tail ++ ] = s;**  **while (front < tail) {**  **for (int i = head[que[front ++ ]]; i != -1; i = e[i].next)**  **if (e[i].c > 0 && dis[e[i].v] == -1){**  **dis[e[i].v] = dis[e[i].u] + 1;**  **if (e[i].v == t)**  **return 1;**  **que[tail ++ ] = e[i].v;**  **}**  **}**  **return 0;**  **}**  **int dinic(int s, int t, int n) {**  **int maxflow = 0, tp;**  **while (bfs(s, t, n)) {**  **for (int i = 0; i < n; i ++ )**  **cur[i] = head[i];**  **int u = s, tail = 0;**  **while(cur[s] != -1){**  **if(u == t){**  **tp = INF;**  **for (int i = tail - 1; i >= 0; i -- )**  **tp = min(tp, e[sta[i]].c);**  **maxflow += tp;**  **for (int i = tail - 1; i >= 0; i -- ){**  **e[sta[i]].c -= tp;**  **e[sta[i] ^ 1].c += tp;**  **if (e[sta[i]].c == 0)**  **tail = i;**  **}**  **u = e[sta[tail]].u;**  **}**  **else if (cur[u] != -1 && e[cur[u]].c > 0**  **&& dis[u] + 1 == dis[e[cur[u]].v]) {**  **sta[tail ++ ] = cur[u];**  **u = e[cur[u]].v;**  **}**  **else {**  **while (u != s && cur[u] == -1)**  **u = e[sta[ -- tail]].u;**  **cur[u] = e[cur[u]].next;**  **}**  **}**  **}**  **return maxflow;**  **}** |

**1.8.3 SAP**

|  |
| --- |
| **/\*SAP\*/**  **int dis[Maxn], cur[Maxn], gap[Maxn], pre[Maxn];**  **int SAP(int s, int t, int n) {**  **/\*可预先加dinic的BFS减少部分时间复杂度\*/**  **memset(dis, 0, sizeof(dis));**  **memset(cur, 0, sizeof(cur));**  **for (int i = 0; i < n; i ++ )**  **cur[i] = head[i];**  **memset(gap, 0, sizeof(gap));**  **int v = s, maxflow = 0;**  **gap[0] = n;**  **while (dis[s] <= n) {**  **bool flag = false;**  **for (int i = cur[v]; i != -1; i = e[i].next)**  **if (e[i].c > 0 && dis[v] == dis[e[i].v] + 1) {**  **flag = true;**  **pre[e[i].v] = v;**  **cur[v] = i;**  **v = e[i].v;**  **break;**  **}**  **if (flag) {**  **if (v == t) {**  **int det = INF;**  **for (int i = v; i != s; i = pre[i])**  **det = min(det, e[cur[pre[i]]].c);**  **for (int i = v; i != s; i = pre[i]) {**  **e[cur[pre[i]]].c -= det;**  **e[cur[pre[i]] ^ 1].c += det;**  **}**  **maxflow += det;**  **v = s;**  **}**  **}**  **else {**  **int mind = n;**  **for (int i = head[v]; i != -1; i = e[i].next )**  **if (e[i].c > 0 && dis[e[i].v] < mind) {**  **mind = dis[e[i].v];**  **cur[v] = i;**  **}**  **if (( -- gap[dis[v]]) == 0) break;**  **gap[dis[v] = mind + 1] ++ ;**  **if (v != s) v = pre[v];**  **}**  **}**  **return maxflow;**  **}** |

**1.9全局最小割**

|  |
| --- |
| **/\*全局最小割**  **\* poj\_2914\*/**  **int f[Maxn][Maxn], maxf[Maxn], v[Maxn];**  **bool vis[Maxn];**  **int Stoer\_Wagner(int n) {**  **int ret = INF;**  **for (int i = 0; i < n; i ++ )**  **v[i] = i;**  **while (n > 1) {**  **int k, s = 0;**  **memset(vis, 0, sizeof(vis));**  **memset(maxf, 0, sizeof(maxf));**  **for (int i = 1; i < n; i ++ ) {**  **k = -1;**  **for (int j = 1; j < n; j ++ )**  **if (!vis[v[j]]) {**  **maxf[v[j]] += f[v[s]][v[j]];**  **if (k == -1 || maxf[v[k]] < maxf[v[j]])**  **k = j;**  **}**  **vis[v[k]] = true;**  **if (i == n - 1) {**  **ret = min(ret, maxf[v[k]]);**  **for (int j = 0; j < n; j ++ ) {**  **f[v[s]][v[j]] += f[v[j]][v[k]];**  **f[v[j]][v[s]] += f[v[j]][v[k]];**  **}**  **v[k] = v[ -- n];**  **}**  **s = k;**  **}**  **}**  **return ret;**  **}** |

**1.10最小费用最大流**

**1.10.1 ZKW费用流**

|  |
| --- |
| **/\* ZKW费用流**  **\* 二分图和稠密图中速度相当快\*/**  **struct edge {**  **int u, v, next, c, w;**  **edge(int u,int v,int next,int c,int w)**  **:u(u),v(v),next(next),c(c),w(w){}**  **edge(){}**  **} e[Maxm];**  **int cnt, head[Maxn], dist[Maxn], cur[Maxn];**  **bool vis[Maxn];**  **void addedge(int u, int v, int c, int w) {**  **e[cnt] = edge(u, v, head[u], c, w);**  **head[u] = cnt ++ ;**  **e[cnt] = edge(v, u, head[v], 0, -w);**  **head[v] = cnt ++ ;**  **}**  **bool modlabel(int n) {**  **int tmp = INF;**  **for (int u = 0; u <= n; u ++ )**  **if (vis[u]) {**  **for (int i = head[u]; i != -1; i = e[i].next) {**  **int v = e[i].v;**  **if (e[i].c && !vis[v])**  **tmp = min(tmp, dist[v] + e[i].w - dist[u]);**  **}**  **}**  **if (tmp == INF) return true;**  **for (int i = 0; i <= n; i ++ )**  **if (vis[i]) {**  **vis[i] = false;**  **dist[i] += tmp;**  **}**  **return false;**  **}**  **int extend(int u, int t, int f) {**  **if (u == t) return f;**  **vis[u] = true;**  **for (int i = cur[u]; i != -1; i = e[i].next) {**  **int v = e[i].v;**  **if (e[i].c && !vis[v] && dist[u] == dist[v] + e[i].w)**  **if (int tmp = extend(v, t, min(f, e[i].c))) {**  **e[i].c -= tmp;**  **e[i ^ 1].c += tmp;**  **cur[u] = i;**  **return tmp;**  **}**  **}**  **return 0;**  **}**  **int mcf(int s, int t, int n) {**  **int mincost, maxflow, tmp;**  **mincost = maxflow = 0;**  **while (1) {**  **for (int i = 0; i <= n; i ++ )**  **cur[i] = head[i];**  **while ((tmp = extend(s, t, INF))) {**  **maxflow += tmp;**  **mincost += tmp \* dist[s];**  **memset(vis, 0, sizeof(vis));**  **}**  **if (modlabel(n)) break;**  **}**  **//return mincost;**  **return maxflow;**  **}** |

**1.10.2一般最小费用最大流**

|  |
| --- |
| **/\*最小费用最大流\*/**  **struct edge {**  **int u, v, next, c, w;**  **edge(int u,int v,int next,int c,int w)**  **:u(u),v(v),next(next),c(c),w(w){}**  **edge(){}**  **} e[Maxm];**  **int cnt, head[Maxn], dist[Maxn], pre[Maxn];**  **bool vis[Maxn];**  **void addedge(int u, int v, int c, int w) {**  **e[cnt] = edge(u, v, head[u], c, w);**  **head[u] = cnt ++ ;**  **e[cnt] = edge(v, u, head[v], 0, -w);**  **head[v] = cnt ++ ;**  **}**  **void init() {**  **memset(head, -1, sizeof(head));**  **cnt = 0;**  **}**  **bool spfa(int s, int t, int n) {**  **memset(dist, 0x3f, sizeof(dist[0]) \* (n + 1));**  **memset(vis, 0, sizeof(vis[0]) \* (n + 1));**  **memset(pre, -1, sizeof(pre[0]) \* (n + 1));**  **queue<int> q;**  **while (!q.empty()) q.pop();**  **q.push(s);**  **vis[s] = true;**  **dist[s] = 0;**  **pre[s] = -1;**  **while (!q.empty()) {**  **int u = q.front();**  **vis[u] =false;**  **q.pop();**  **for (int i = head[u]; i != -1; i = e[i].next)**  **if (e[i].c > 0 && dist[u] + e[i].w < dist[e[i].v]) {**  **dist[e[i].v] = dist[u] + e[i].w;**  **pre[e[i].v] = i;**  **if (!vis[e[i].v]) {**  **q.push(e[i].v);**  **vis[e[i].v] = true;**  **}**  **}**  **}**  **if (dist[t] == INF) return false;**  **else return true;**  **}**  **int fill(int t) {**  **int det = INF, u = t;**  **while (~pre[u]) {**  **u = pre[u];**  **det = min(det, e[u].c);**  **u = e[u].u;**  **}**  **u = t;**  **while (~pre[u]) {**  **u = pre[u];**  **e[u].c -= det;**  **e[u ^ 1].c += det;**  **u = e[u].u;**  **}**  **return det;**  **}**  **int mcf(int s, int t, int n) {**  **int mincost, maxflow;**  **mincost = maxflow = 0;**  **while (spfa(s, t, n)) {**  **int det = fill(t);**  **mincost += det \* dist[t];**  **maxflow += det;**  **}**  **//return mincost;**  **return maxflow;**  **}** |

**1.11二分图最大匹配**

|  |
| --- |
| **/\*二分图最大匹配\*/**  **bool find(int u, int root) {**  **for (int i = head[u]; i != -1; i = e[i].next) {**  **int v = e[i].v;**  **if (vis[v] == root) continue;**  **vis[v] = root;**  **if (next[v] == -1 || find(next[v], root)) {**  **next[v] = u;**  **return true;**  **}**  **}**  **return false;**  **}**  **int work() {**  **int cnt1 = 0;**  **memset(vis, 0, sizeof(vis));**  **for (int i = 0; i < n; i ++ )**  **if (find(id[i], i + 1)) cnt1 ++ ;**  **return cnt1;**  **}** |

**1.12欧拉回路**

|  |
| --- |
| **/\*欧拉回路\*/**  **VI ve[Maxn];**  **int cur[Maxn];**  **stack<int> eulerianWalk(int u) { //返回欧拉回路的逆序**  **stack<int> sta, ret;**  **sta.push(u);**  **cur[u] = 0;**  **while (!sta.empty()) {**  **u = sta.top();**  **sta.pop();**  **while (cur[u] < ve[u].size()) {**  **sta.push(u);**  **u = ve[u][cur[u] ++ ];**  **}**  **ret.push(u);**  **}**  **return ret;**  **}** |

**1.13一般图最大匹配**

|  |
| --- |
| **#define Maxm 250 \* 250 \* 2**  **#define Maxn 250**  **deque<int> Q;**  **//g[i][j]存放关系图：i,j是否有边,match[i]存放i所匹配的点**  **bool g[Maxn][Maxn], inque[Maxn], inblossom[Maxn];**  **int match[Maxn], pre[Maxn], base[Maxn];**  **//找公共祖先**  **int findancestor(int u,int v){**  **bool inpath[Maxn] = {false};**  **while(1) {**  **u = base[u];**  **inpath[u] = true;**  **if(match[u] == -1) break;**  **u = pre[match[u]];**  **}**  **while(1) {**  **v = base[v];**  **if(inpath[v]) return v;**  **v = pre[match[v]];**  **}**  **}**  **//压缩花**  **void reset(int u,int anc) {**  **while(u != anc) {**  **int v = match[u];**  **inblossom[base[u]] = 1;**  **inblossom[base[v]] = 1;**  **v = pre[v];**  **if(base[v] != anc)pre[v] = match[u];**  **u = v;**  **}**  **}**  **void contract(int u, int v, int n) {**  **int anc = findancestor(u, v);**  **//SET(inblossom,0);**  **memset(inblossom, 0, sizeof(inblossom));**  **reset(u, anc);**  **reset(v, anc);**  **if(base[u] != anc) pre[u] = v;**  **if(base[v] != anc) pre[v] = u;**  **for(int i = 1; i <= n; i ++ )**  **if(inblossom[base[i]]) {**  **base[i] = anc;**  **if(!inque[i]) {**  **Q.push\_back(i);**  **inque[i] = 1;**  **}**  **}**  **}**  **bool dfs(int S, int n) {**  **for(int i = 0; i <= n; i ++ )**  **pre[i] = -1, inque[i] = 0, base[i] = i;**  **Q.clear();**  **Q.push\_back(S);**  **inque[S] = 1;**  **while(!Q.empty()){**  **int u = Q.front();**  **Q.pop\_front();**  **for(int v = 1; v <= n; v ++ ) {**  **if(g[u][v] && base[v] != base[u] && match[u] != v) {**  **if(v == S || (match[v] != -1 && pre[match[v]] != -1))**  **contract(u,v,n);**  **else if(pre[v] == -1) {**  **pre[v] = u;**  **if(match[v] != -1)**  **Q.push\_back(match[v]), inque[match[v]] = 1;**  **else {**  **u = v;**  **while(u != -1) {**  **v = pre[u];**  **int w = match[v];**  **match[u] = v;**  **match[v] = u;**  **u = w;**  **}**  **return true;**  **}**  **}**  **}**  **}**  **}**  **return false;**  **}**  **int main() {**  **int n, m, a, b, ans;**  **while(scanf("%d", &n) != EOF) {**  **ans = 0; //最多有几对匹配**  **memset(match, -1, sizeof(match));**  **memset(g, 0, sizeof(g));**  **while(scanf("%d%d",&a,&b) != EOF && a != 0) {**  **g[a][b] = g[b][a] = 1;**  **}**  **for(int i = 1; i <= n; i ++ ) {**  **if(match[i] == -1 && dfs(i, n)){**  **ans ++ ;**  **}**  **}**  **cout<<ans \* 2<<endl;**  **for(int i = 1; i <= n; i ++ ) {**  **if(match[i] != -1){**  **printf("%d %d\n", i, match[i]);**  **match[i] = match[match[i]] = -1;**  **}**  **}**  **}**  **return 0;**  **}** |

**1.14 DLX**

|  |
| --- |
| **/\* dancing link**  **\* 精确覆盖问题**  **\* 可以添加迭代加深优化：**  **\* 1）枚举深度h；**  **\* 2）若当前深度+predeep > h return false；**  **\***  **int predeep() {**  **bool vis[Maxm];**  **memset(vis, 0, sizeof(vis));**  **int ret = 0;**  **for (Node \*p = head->R; p != head; p = p->R)**  **if (!vis[p->col]) {**  **ret ++ ;**  **vis[p->col] ++ ;**  **for (Node \*q = p->D; q != p; q = p->D)**  **for (Node \*r = q->R; r != q; r = r->R)**  **vis[r->col] = true;**  **}**  **return ret;**  **}**  **\* \*/**  **#define Maxn 1010**  **#define Maxm 1010**  **struct Node {**  **Node \*L, \*R, \*U, \*D;**  **int col, row;**  **} \*head, \*row[Maxn], \*col[Maxm], node[Maxn \* Maxm];**  **int colsum[Maxm], cnt;**  **void init(int mat[][Maxm], int n, int m) {**  **cnt = 0;**  **memset(colsum, 0, sizeof(colsum));**  **head = &node[cnt ++ ];**  **for (int i = 1; i <= n; i ++ )**  **row[i] = &node[cnt ++ ];**  **for (int j = 1; j <= m; j ++ )**  **col[j] = &node[cnt ++ ];**    **head->D = row[1], row[1]->U = head;**  **head->R = col[1], col[1]->L = head;**  **head->U = row[n], row[n]->D = head;**  **head->L = col[m], col[m]->R = head;**  **head->row = head->col = 0;**  **for (int i = 1; i <= n; i ++ ) {**  **if (i != n) row[i]->D = row[i + 1];**  **if (i != 1) row[i]->U = row[i - 1];**  **row[i]->L = row[i]->R = row[i];**  **row[i]->row = i, row[i]->col = 0;**  **}**  **for (int i = 1; i <= m; i ++ ) {**  **if (i != m) col[i]->R = col[i + 1];**  **if (i != 1) col[i]->L = col[i - 1];**  **col[i]->U = col[i]->D = col[i];**  **col[i]->col = i, col[i]->row = 0;**  **}**  **for (int i = n; i > 0; i -- )**  **for (int j = m; j > 0; j -- )**  **if (mat[i][j]) {**  **Node \*p = &node[cnt ++ ];**  **p->R = row[i]->R, row[i]->R->L = p;**  **p->L = row[i], row[i]->R = p;**  **p->D = col[j]->D, col[j]->D->U = p;**  **p->U = col[j], col[j]->D = p;**  **p->row = i;**  **p->col = j;**  **colsum[j] ++ ;**  **}**  **}**  **/\*多重覆盖只需删除列，无需对应行删除**  **void remove(Node \*c) {**  **for (Node \*p = c->D; p != c; p = p->D) {**  **p->L->R = p->R;**  **p->R->L = p->L;**  **}**  **}**  **\*/**  **void remove(Node \*c) {**  **c->L->R = c->R;**  **c->R->L = c->L;**  **for (Node \*p = c->D; p != c; p = p->D) {**  **for (Node \*q = p->R; q != p; q = q->R) {**  **q->U->D = q->D;**  **q->D->U = q->U;**  **colsum[q->col] -- ;**  **}**  **}**  **}**  **void resume(Node \*c) {**  **for (Node \*p = c->U; p != c; p = p->U) {**  **for (Node \*q = p->L; q != p; q = q->L) {**  **q->U->D = q;**  **q->D->U = q;**  **colsum[q->col] ++ ;**  **}**  **}**  **col[c->col]->L->R = col[c->col];**  **col[c->col]->R->L = col[c->col];**  **}**  **int ans[Maxm];**  **int dfs(int deep) {**  **if (head->R == head) return deep;**  **Node \*p, \*q = head->R;**  **for (p = head->R; p != head; p = p->R)**  **if (colsum[p->col] < colsum[q->col])**  **q = p;**  **remove(q);**  **for (p = q->D; p != q; p = p->D) {**  **for (Node\* r = p->R; r != p; r = r->R)**  **if (r->col != 0)**  **remove(col[r->col]);**  **/\*--------可修改区域-----------\*/**  **ans[deep] = p->row;**  **/\*-----------------------------\*/**  **int sta = dfs(deep + 1);**  **if (sta != -1) return sta;**  **for (Node\* r = p->L; r != p; r = r->L)**  **if (r->col != 0)**  **resume(col[r->col]);**  **}**  **resume(q);**  **return -1;**  **}** |

****2 计算几何****

**2.1平面几何**

|  |
| --- |
| **int dcmp(double x) {return x < -eps ? -1 : x > eps;}**  **struct point {**  **double x, y;**  **point() {}**  **point(double \_x, double \_y) : x(\_x), y(\_y) {}**  **point operator - (point p) {return point(x - p.x, y - p.y);}**  **point operator + (point p) {return point(x + p.x, y + p.y);}**  **point operator / (double d){return point(x / d, y / d);}**  **point operator \* (double d){return point(x \* d, y \* d);}**  **bool operator < (const point &p)const {**  **return dcmp(x - p.x) == 0 ?**  **dcmp(y - p.y) <= 0 : dcmp(x - p.x) <= 0;**  **}**  **double operator \* (point p) {return x \* p.y - y \* p.x;} //叉积**  **double operator | (point p) {return x \* p.x + y \* p.y;} //点积**  **double len(){return hypot(x, y);}**  **double arc(){return atan2(y, x);}**  **point normal() {return (\*this) / this->len();}**  **point rotate(){return point(-y, x);}**  **point rotate(double arc){**  **return point(x \* cos(arc) - y \* sin(arc),**  **x \* sin(arc) + y \* cos(arc));**  **}**  **double dis(point p){return (\*this - p).len();}**  **double dis2(point p){p = p - (\*this); return p|p;}**  **void in() {scanf("%lf%lf", &x, &y);}**  **void out(){printf("%f %f\n",x,y);}**  **};**  **const point O = point(0, 0);**  **double rad(point p1, point p2) {**  **double thi1 = fabs(p1.arc() - p2.arc());**  **return thi1 > pi ? pi \* 2 - thi1 : thi1;**  **}**  **/\*点到直线的距离\*/**  **double disLP(point p1, point p2, point q) {**  **return fabs((p1 - p2).normal() \* (q - p1));**  **}**  **/\*点到线段的距离\*/**  **double disSP(point p1, point p2, point q) {**  **if (dcmp((p2 - p1) | (q - p1)) <= 0) return q.dis(p1);**  **if (dcmp((p1 - p2) | (q - p2)) <= 0) return q.dis(p2);**  **return disLP(p1, p2, q);**  **}**  **/\*直线和直线求交点\*/**  **bool isLL(point p1, point p2, point q1, point q2, point &is) {**  **double m = (q2 - q1) \* (p1 - q1), n = (q2 - q1) \* (p2 - q1);**  **if (dcmp(n - m) == 0) return 0; //判直线平行，n=m=0时直线重合**  **is = (p1 \* n - p2 \* m) / (n - m);**  **return 1;**  **}**  **/\*点在线段上\*/**  **int crsSP(point p1, point p2, point q) {**  **return dcmp(disSP(p1, p2, q)) != 0 ?**  **-1 : dcmp((p1 - q) | (p2 - q)) != 0;**  **}**  **/\*线段和线段是否相交, 非规范相交\*/**  **bool crsSS(point p1, point p2, point q1, point q2) {**  **if (dcmp(min(q1.x, q2.x) - max(p1.x, p2.x)) > 0) return false;**  **if (dcmp(min(p1.x, p2.x) - max(q1.x, q2.x)) > 0) return false;**  **if (dcmp(min(q1.y, q2.y) - max(p1.y, p2.y)) > 0) return false;**  **if (dcmp(min(p1.y, p2.y) - max(q1.y, q2.y)) > 0) return false;**  **int c1 = dcmp((p2 - p1) \* (q1 - p1)),**  **c2 = dcmp((p2 - p1) \* (q2 - p1)),**  **c3 = dcmp((q2 - q1) \* (p1 - q1)),**  **c4 = dcmp((q2 - q1) \* (p2 - q1));**  **return c1 \* c2 <= 0 && c3 \* c4 <= 0;**  **}**  **/\*线段与线段求交，规范相交\*/**  **// bool crsSS(point p1, point p2, point q1, point q2) {**  **// int c1 = dcmp((p2 - p1) \* (q1 - p1)),**  **// c2 = dcmp((p2 - p1) \* (q2 - p1)),**  **// c3 = dcmp((q2 - q1) \* (p1 - q1)),**  **// c4 = dcmp((q2 - q1) \* (p2 - q1));**  **// return c1 \* c2 < 0 && c3 \* c4 < 0;**  **// }**  **/\*判断圆与线段是否相交\*/**  **bool crsCS(point o, double r, point p1, point p2) {**  **return dcmp(disSP(p1, p2, o) - r) <= 0 &&**  **(dcmp(p1.dis(o) - r) >= 0 || dcmp(p2.dis(o) - r) >= 0);**  **}**  **/\*点到直线的垂足\*/**  **point proj(point p1, point p2, point q) {**  **point d = p2 - p1;**  **return p1 + d \* (d | (q - p1)) / p1.dis2(p2);**  **}**  **/\*直线与圆的交点, p1 != p2, 返回值中r1->r2 与 p1->p2同向\*/**  **vector<point> isCL(point o, double r, point p1, point p2) {**  **vector<point> ret;**  **point d = p1 - p2, is;**  **isLL(p1, p2, o, o + d.rotate(), is);**  **double val = is.dis(o);**  **val = r \* r - val \* val;**  **if (dcmp(val) == 0) ret.push\_back(is);**  **else**  **if (dcmp(val) > 0) {**  **val = sqrt(val) / d.len();**  **ret.push\_back(is + d \* val);**  **ret.push\_back(is - d \* val);**  **}**  **return ret;**  **}**  **/\*圆与圆的交点\*/**  **vector<point> isCC(point o1, double r1, point o2, double r2) {**  **vector<point> ret;**  **double x = o1.dis2(o2);**  **double y = ((r1 \* r1 - r2 \* r2) / x + 1) / 2;**  **double d = r1 \* r1 / x - y \* y;**  **if (dcmp(d) >= 0) {**  **point q1 = (o1 + (o2 - o1) \* y);**  **if (dcmp(d) == 0)**  **ret.push\_back(q1);**  **else {**  **point q2 = ((o2 - o1) \* sqrt(d)).rotate();**  **ret.push\_back(q1 + q2);**  **ret.push\_back(q1 - q2);**  **}**  **}**  **return ret;**  **}**  **/\*点到圆的切线\*/**  **vector<point> tanCP(point o, double r, point p) {**  **vector<point> ret;**  **double x = p.dis2(o);**  **double d = x - r \* r;**  **if (dcmp(d) >= 0) {**  **if (dcmp(d) == 0)**  **ret.push\_back(p);**  **else {**  **point q1 = (p - o) \* (r \* r / x);**  **point q2 = ((p - o) \* (- r \* sqrt(d) / x)).rotate();**  **ret.push\_back(o + (q1 - q2));**  **ret.push\_back(o + (q1 + q2));**  **}**  **}**  **return ret;**  **}**  **/\*两圆的共同切线\*/**  **vector<pair<point, point> > tanCC(point o1, double r1, point o2, double r2) {**  **vector<pair<point, point> > line;**  **vector<point> to1, to2;**  **if (fabs(r1 - r2) < eps) {**  **point dir = o2 - o1;**  **dir = (dir.normal() \* r1).rotate();**  **line.PB(MP(o1 + dir, o2 + dir));**  **line.PB(MP(o1 - dir, o2 - dir));**  **}**  **else {**  **point p = (o2 \* r1 - o1 \* r2) / (r1 - r2);**  **to1 = tanCP(o1, r1, p);**  **to2 = tanCP(o2, r2, p);**  **for (int i = 0; i < to1.size() && i < to2.size(); i ++ )**  **line.PB(MP(to1[i], to2[i]));**  **}**  **point p = (o1 \* r2 + o2 \* r1) / (r1 + r2);**  **to1 = tanCP(o1, r1, p);**  **to2 = tanCP(o2, r2, p);**  **for (int i = 0; i < to1.size() && i < to2.size(); i ++ )**  **line.PB(MP(to1[i], to2[i]));**  **return line;**  **}**  **/\*两圆的面积交\*/**  **double areaCC(point o1, double r1, point o2, double r2) {**  **double d = o1.dis(o2);**  **if (dcmp(r1 + r2 - d) <= 0) return 0;**  **if (dcmp(d - fabs(r1 - r2)) <= 0) {**  **double r = min(r1, r2);**  **return pi \* r \* r;**  **}**  **double x = (d \* d + r1 \* r1 - r2 \* r2) / (2 \* d);**  **double t1 = acos(x / r1);**  **double t2 = acos((d - x) / r2);**  **return r1 \* r1 \* t1 + r2 \* r2 \* t2 - d \* r1 \* sin(t1);**  **}**  **/\*圆心在原点o半径为r的圆与三角形o,p1,p2的有向面积\*/**  **double areaCT(double r, point p1, point p2) {**  **vector<point> q = isCL(O, r, p1, p2);**  **if (q.size() == 0) return r \* r \* rad(p1, p2) / 2;**  **if (q.size() == 1) q.push\_back(q[0]);**  **bool b1 = dcmp(p1.len() - r) > 0, b2 = dcmp(p2.len() - r) > 0;**  **if (b1 && b2) {**  **if (dcmp((p1 - q[0]) | (p2 - q[0])) <= 0 &&**  **dcmp((p1 - q[1]) | (p2 - q[1])) <= 0) {**  **return (r \* r \* (rad(p1, p2)-rad(q[0], q[1])) + q[0] \* q[1]) / 2;**  **}**  **else return r \* r \* rad(p1, p2) / 2;**  **}**  **else if (b1) return (r \* r \* rad(p1, q[0]) + q[0] \* p2) / 2;**  **else if (b2) return (r \* r \* rad(q[1], p2) + p1 \* q[1]) / 2;**  **else return p1 \* p2 / 2;**  **}**  **/\*凸多边形\*/**  **/\*求凸包,逆时针\*/**  **vector<point> ConvexHull(vector<point> p) {**  **int n = p.size(), m = 0;**  **vector<point> q;**  **q.resize(n \* 2);**  **sort(p.begin(), p.end());**  **for (int i = 0; i < n; i ++ ) {**  **while (m > 1 && dcmp((q[m - 1] - q[m - 2]) \* (p[i] - q[m - 2])) <= 0)**  **m -- ;**  **q[m ++ ] = p[i];**  **}**  **int k = m;**  **for (int i = n - 2; i >= 0; i -- ) {**  **while (m > k && dcmp((q[m - 1] - q[m - 2]) \* (p[i] - q[m - 2])) <= 0)**  **m -- ;**  **q[m ++ ] = p[i];**  **}**  **if (n > 1) m -- ;**  **q.resize(m);**  **return q;**  **}**  **/\*直线切割多边形, 切割在p1,p2的左侧\*/**  **vector<point> ConvexCut(vector<point> p, point p1, point p2) {**  **vector<point> q;**  **point is;**  **int n = p.size();**  **for (int i = 0; i < n; i ++ ) {**  **int d1 = dcmp((p2 - p1) \* (p[i] - p1));**  **int d2 = dcmp((p2 - p1) \* (p[(i + 1) % n] - p1));**  **if (d1 >= 0) q.push\_back(p[i]);**  **if (d1 \* d2 < 0) {**  **isLL(p1, p2, p[i], p[(i + 1) % n], is);**  **q.push\_back(is);**  **}**  **}**  **return q;**  **}**  **/\*点在多边形里, -1在多边形外，0在多边形上，1在多边形内\*/**  **int InConvex(vector<point> p, point q) {**  **int n = p.size();**  **int res = -1;**  **for (int i = 0; i < n; i ++ ) {**  **point a = p[i] - q, b = p[(i + 1) % n] - q;**  **if (a.y > b.y) swap(a, b);**  **if (dcmp(a.y) <= 0 && dcmp(b.y) > 0 && dcmp(a \* b) > 0) res = - res;**  **if (dcmp(a \* b) == 0 && dcmp(a | b) <= 0) return 0;**  **}**  **return res;**  **}**  **/\*凸包外一点到凸包的最近距离----log(n),**  **\* 构建两侧法线划分出可行区域，凸包需要**  **\* 逆时针\*/**  **bool in(point p1, point p2, point q) {**  **return dcmp(p1 \* q) >= 0 && dcmp(p2 \* q) <= 0;**  **}**  **bool in(point p1, point p2, point p3, point p4, point q) {**  **point o12 = (p1 - p2).rotate();**  **point o23 = (p2 - p3).rotate();**  **point o34 = (p3 - p4).rotate();**  **return in(o12, o23, q - p2) || in(o23, o34, q - p3)**  **|| in(o23, p3 - p2, q - p2) || (in(p2 - p3, o23, q - p3));**  **}**  **double disConvexP(vector<point> p, point q) {**  **int n = p.size();**  **int left = 0, right = n;**  **while (right - left > 1) {**  **int mid = (left + right) / 2;**  **if (in(p[(left + n - 1) % n], p[left], p[mid], p[(mid + 1) % n], q))**  **right = mid;**  **else left = mid;**  **}**  **return disSP(p[left], p[right % n], q);**  **}**  **/\* 凸包的直径,旋转卡壳----O(n)????**  **\* 类似计算两凸包的最近与最远**  **\* 距离\*/**  **double ConvecDiameter(vector<point> p) {**  **int n = p.size(), j = 1;**  **double maxd = 0;**  **p.push\_back(p[0]);**  **for (int i = 0; i < n; i ++ ) {**  **while ((p[i + 1] - p[i]) \* (p[j + 1] - p[i])**  **> (p[i + 1] - p[i]) \* (p[j] - p[i]))**  **j = (j + 1) % n;**  **maxd = max(maxd, max(p[i].dis(p[j]), p[i + 1].dis(p[j + 1])));**  **}**  **return maxd;**  **}**  **/\*半平面\*/**  **struct Line {**  **point u, v;**  **double arc;**  **Line() {}**  **Line(point u, point v) : u(u), v(v) {**  **arc = atan2(v.y - u.y, v.x - u.x);**  **}**  **bool operator < (const Line &p) const {**  **return arc < p.arc;**  **}**  **};**  **bool OnLeft(Line l, point p) {**  **return (l.v - l.u) \* (p - l.u) >= 0;//交集为点时>0，否则>=0**  **}**  **vector<point> crsConvexConvex(vector<Line> l) {**  **vector<point> p;**  **vector<Line> q;**  **int n = l.size(), first, last;**  **sort(l.begin(), l.end());**  **p.resize(n);**  **q.resize(n);**  **q[first = last = 0] = l[0];**  **for (int i = 1; i < n; i ++ ) {**  **while (first < last && !OnLeft(l[i], p[last - 1])) last -- ;**  **while (first < last && !OnLeft(l[i], p[first])) first ++ ;**  **q[ ++ last] = l[i];**  **if (dcmp((q[last].v - q[last].u)**  **\* (q[last - 1].v - q[last - 1].u)) == 0) {**  **last -- ;**  **if (OnLeft(q[last], l[i].u)) q[last] = l[i];**  **}**  **if (first < last)**  **isLL(q[last - 1].u, q[last - 1].v, q[last].u, q[last].v, p[last - 1]);**  **}**  **while (first < last && !OnLeft(q[first], p[last - 1])) last -- ;**  **if (last - first <= 1) {**  **p.clear();**  **return p;**  **}**  **isLL(q[first].u, q[first].v, q[last].u, q[last].v, p[last]);**  **p.resize(last + 1);**  **p.erase(p.begin(), p.begin() + first);**  **return p;**  **}** |

**2.2三维几何**

|  |
| --- |
| **int dcmp(double x) {return x < -eps ? -1 : x > eps;}**  **struct point {**  **double x, y, z;**  **point() {}**  **point(double \_x, double \_y, double \_z) : x(\_x), y(\_y), z(\_z) {}**  **void in() {scanf("%lf%lf%lf",&x,&y,&z);}**  **void out() {printf("%.10f %.10f %.10f\n", x + eps, y + eps, z + eps);}**  **point operator - (point p) {return point(x - p.x, y - p.y, z - p.z);}**  **point operator + (point p) {return point(x + p.x, y + p.y, z + p.z);}**  **point operator \* (point p) {**  **return point(y \* p.z - z \* p.y,**  **z \* p.x - x \* p.z,**  **x \* p.y - y \* p.x);**  **}**  **point operator \* (double d) {return point(x \* d, y \* d, z \* d);}**  **point operator / (double d) {return point(x / d, y / d, z / d);}**  **double operator | (point p) {return x \* p.x + y \* p.y + z \* p.z;}**  **bool operator < (const point &p) const {**  **return dcmp(x - p.x) == 0 ?**  **(dcmp(y - p.y) == 0 ? dcmp(z - p.z) <= 0**  **: dcmp(y - p.y) <= 0)**  **: dcmp(x - p.x) <= 0;**  **} point normal() {return (\*this) / this->len();}**  **double len() {return sqrt(x \* x + y \* y + z \* z);}**  **double dis(point p) {return (\*this-p).len();}**  **double dis2(point p) {p = p - (\*this); return p | p;}**  **};**  **const point O = point(0, 0, 0);**  **const point PINF = point(-inf, -inf, -inf);**  **/\*点到直线的距离\*/**  **double disLP(point p1, point p2, point q) {**  **return ((p2 - p1) \* (q - p1)).len() / (p2 - p1).len();**  **}**  **/\*直线与直线的距离\*/**  **double disLL(point p1, point p2, point q1, point q2) {**  **point p = q1 - p1;**  **point u = p2 - p1;**  **point v = q2 - q1;**  **double d = u.dis2(O) \* v.dis2(O) - (u | v) \* (u | v);**  **if (dcmp(d) == 0) return disLP(q1, q2, p1);**  **double s = ((p | u) \* v.dis2(O) - (p | v) \* (u | v)) / d;**  **return disLP(q1, q2, p1 + u \* s);**  **}**  **/\* 平面与直线的交点**  **\* 无交点返回无穷远点\*/**  **point isFL(point p, point o, point q1, point q2) {**  **double a = o | (q2 - p);**  **double b = o | (q1 - p);**  **double d = a - b;**  **if (dcmp(d) == 0) return PINF;**  **return (q1 \* a - q2 \* b) / d;**  **}**  **/\*平面与平面的交线\*/**  **vector<point> isFF(point p1, point o1, point p2, point o2) {**  **vector<point> p;**  **point e = o1 \* o2;**  **point v = o1 \* e;**  **double d = o2 | v;**  **if (dcmp(d) == 0) return p;**  **point q = p1 + v \* ((o2 | (p2 - p1)) / d);**  **p.push\_back(q);**  **p.push\_back(q + e);**  **return p;**  **}**  **/\* 三维凸包\*/**  **#define Maxn 1010**  **vector<point> p;**  **struct face {**  **int v[3];**  **bool flag;**  **int cansee(int i) const {**  **return ((p[i] - p[v[0]]) | ((p[v[1]] - p[v[0]])**  **\* (p[v[2]] - p[v[0]]))) > eps;**  **}**  **bool sameface(face f) const {**  **for (int i = 0; i < 3; i ++ )**  **if (dcmp((p[f.v[i]] - p[v[0]]) | ((p[v[1]] - p[v[0]])**  **\* (p[v[2]] - p[v[0]]))) != 0)**  **return false;**  **return true;**  **}**  **};**  **int lin[Maxn][Maxn];**  **const int nxt[3] = {1, 2, 0};**  **vector<face> cur;**  **void dfs(int j, int t) {**  **face add;**  **cur[j].flag = false;**  **for (int i = 0; i < 3; i ++ ) {**  **int tmp = lin[cur[j].v[nxt[i]]][cur[j].v[i]];**  **if (cur[tmp].flag) {**  **if (cur[tmp].cansee(t)) dfs(tmp, t);**  **else {**  **add = (face){{cur[j].v[i], cur[j].v[nxt[i]], t}, true};**  **cur.push\_back(add);**  **for (int k = 0; k < 3; k ++ )**  **lin[add.v[k]][add.v[nxt[k]]] = cur.size() - 1;**  **}**  **}**  **}**  **}**  **vector<face> convexHull() {**  **int n = p.size();**  **vector<face> ans;**  **/\*------------确保前4点不共面--------------\*/**  **bool ck = true;**  **for (int i = 1; i < n; i ++ )**  **if (p[0].dis(p[i]) > eps) {**  **swap(p[i], p[1]);**  **ck = false;**  **break;**  **}**  **if (ck) return ans;**  **ck = true;**  **for (int i = 2; i < n; i ++ )**  **if (((p[1] - p[0]) \* (p[i] - p[0])).len() > eps) {**  **swap(p[i], p[2]);**  **ck = false;**  **break;**  **}**  **if (ck) return ans;**  **ck = true;**  **for (int i = 3; i < n; i ++ )**  **if (fabs((p[i] - p[0]) | ((p[1] - p[0]) \* (p[2] - p[0]))) > eps) {**  **swap(p[i], p[3]);**  **ck = false;**  **break;**  **}**  **if (ck) return ans;**  **/\*----------------------------------------\*/**  **cur.clear();**  **for (int i = 0; i < 4; i ++ ) {**  **cur.push\_back((face){{(i + 1) % 4, (i + 2) % 4, (i + 3) % 4}, true});**  **if (cur[i].cansee(i))**  **swap(cur[i].v[1], cur[i].v[2]);**  **for (int j = 0; j < 3; j ++ )**  **lin[cur[i].v[j]][cur[i].v[nxt[j]]] = i;**  **}**  **for (int i = 4; i < n; i ++ )**  **for (int j = 0; j < cur.size(); j ++ )**  **if (cur[j].flag && cur[j].cansee(i)) {**  **dfs(j, i);**  **break;**  **}**  **for (int i = 0; i < cur.size(); i ++ )**  **if (cur[i].flag)**  **ans.push\_back(cur[i]);**  **return ans;**  **}**  **bool vis[Maxn \* 4];**  **int faceCountDfs(const vector<face> &f, int t) {**  **if (vis[t]) return 0;**  **vis[t] = true;**  **int ret = 1;**  **for (int i = 0; i < 3; i ++ ) {**  **int s = lin[f[t].v[nxt[i]]][f[t].v[i]];**  **if (vis[s]) continue;**  **ret += faceCountDfs(f, s);**  **if (f[t].sameface(f[s])) ret -- ;**  **}**  **return ret;**  **}**  **int faceCount(const vector<face> &f) {**  **memset(vis, 0, sizeof(vis));**  **for (int i = 0; i < f.size(); i ++ )**  **for (int j = 0; j < 3; j ++ )**  **lin[f[i].v[j]][f[i].v[nxt[j]]] = i;**  **return faceCountDfs(f, 0);**  **}** |

**2.3相关公式**

**2.3.1三角形相关**

**内心：**

**外心：**

垂心：

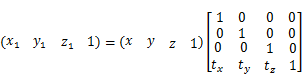
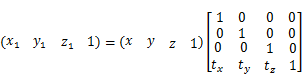
外接圆半径：

海伦公式：

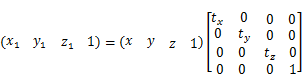
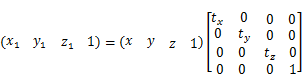
PICK公式：

**2.3.2三维旋转**

平移：



比例变换：



绕x轴旋转：



绕y轴旋转：

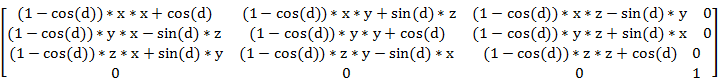


绕z轴旋转：



绕任意轴旋转：

**给定具有单位长的旋转轴**和旋转角度(逆时针方向)，则绕OA轴旋转变换的矩阵表示可确定为：



**2.3.3角度相关**

**2.3.4 四面体相关**

**六边长分别为a,b,c,d,e,f四面体体积**



****3数学****

**3.1行列式求值**

|  |
| --- |
| /\*注意：1、mod可以任意；2、可用于行列式求值，将循环i改为从0开始即可。\*/  LL det(LL a[][N],int n) {//矩阵a[0~n-1],大小n  for (int i = 0; i < n; i ++ )  for (int j = 0; j < n; j ++ )  a[i][j] %= mod;  long long ret = 1;  for (int i = 1; i < n; i ++ ) {  for (int j = i + 1; j < n; j ++ )  while (a[j][i]) {  LL t = a[i][i] / a[j][i];  for (int k = i; k < n; k ++ )  a[i][k] = (a[i][k] - a[j][k] \* t) % mod;  for (int k = i; k < n; k ++ )  swap(a[i][k], a[j][k]);  ret = -ret;  }  if(a[i][i] == 0)return 0;  ret = ret \* a[i][i] % mod;  }  return (ret + mod) % mod;  } |

**3.2高斯消元法**

**3.2.1异或方程组**

|  |
| --- |
| const int sizeM = 110;  const int sizeN = 110;  const int MOD = 1000007;  struct Matrix {  int m, n; //m个方程，n个未知数，1~n  int mat[sizeM][sizeN];  };  /\*返回不相关的方程组个数\*/  int Guass(struct Matrix &a) {  int row = a.m, col = a.n + 1;  int k;  int s = a.n;  for (k = 1; k <= a.n; k++) {  int x = -1;  do {  for (int i = k; i <= row; i++) //找第k列的非0元  if (a.mat[i][k]) {  x = i;  break;  }  if (x == -1) {  for (int i = 1; i <= row; i++)  swap(a.mat[i][k], a.mat[i][s]);  s --;  }  } while (x == -1 && s >= k);  if (x == -1) {  k --;  break;  }  if (x != k)//行交换  for (int j = k; j <= col; j++)  swap(a.mat[x][j], a.mat[k][j]);  for (int i = 1; i <= row; i++)//其他行与第k行异或  if (i != k && a.mat[i][k]) {  for (int j = k; j <= col; j++)  a.mat[i][j] = (a.mat[i][j] ^ a.mat[k][j]);  }  }  if (k > a.n) k --;  return k;  } |

**3.2.2普通方程组**

|  |
| --- |
| const int sizeM = 110;  const int sizeN = 110;  const int MOD = 1000007;  struct Matrix {  double mat[sizeM][sizeN];  int m, n; //m个方程，n个未知数  };  int Guass(struct Matrix &a) {  int row = a.m, col = a.n + 1;  int k;  int s = a.n;  for (k = 1; k <= a.n; k++) {  double maxp= 0.0;  int x = -1;  do {  for (int i = k; i <= row; i++) //找第k列的非0元  if (fabs(a.mat[i][k]) > fabs(maxp)) {  maxp = a.mat[i][k];  x = i;  break;  }  if (x == -1) {  for (int i = 1; i <= row; i++)  swap(a.mat[i][k], a.mat[i][s]);  s --;  }  } while (x == -1 && s >= k);  if (x == -1 || fabs(maxp) <= 1e-8) {  k --;  break;  }  //行交换  if (x != k)  for (int j = k; j <= col; j++)  swap(a.mat[x][j], a.mat[k][j]);  //第 k 行同a.mat[k][k]  for (int j = col; j >= k; j--)  a.mat[k][j] /= a.mat[k][k];  //其他行与第k行相减  for (int i = 1; i <= row; i++)  if (i != k && fabs(a.mat[i][k]) > 0.0) {  for (int j = col; j >= k; j--)  a.mat[i][j] = (a.mat[i][j] - a.mat[k][j] \* a.mat[i][k]);  }  }  if (k > a.n) k --;  return k;  } |

**3.3组合数学相关定理**

**3.3.1各种数列**

**3.3.2五边形数定理（hdu\_4651,4658）**

五边形数:1,2,5,7,12,15,22,25…

欧拉函数展开式



与分割函数之间的关系

欧拉函数的倒数是分割函数的母函数





**3.3.3 莫比乌斯反演（hdu\_4746）**



3.3.4 Burnside引理

设置换群G作用于有限集合上，则在G作用下的等价类的数目为：



其中为g在上的不动点个数，即满足的个数.



3.3.5 Polya定理

设G是p个对象的一个置换群，用k种颜色涂染这p个对象，若一种染色方案在群G的作用下变成另外一种方案，则这两种方案当做同一种方案，这样不同染色方案数为：



其中为置换f的循环节。



****4比赛头文件****

|  |
| --- |
| #include <cstdlib>  #include <cctype>  #include <cstring>  #include <cstdio>  #include <cmath>  #include <algorithm>  #include <vector>  #include <string>  #include <iostream>  #include <map>  #include <set>  #include <queue>  #include <list>  using namespace std;  #define PB push\_back  #define MP make\_pair  #define A first  #define B second  #define eps 1e-8  #define INF 0x3f3f3f3f  #define inf 1e20  typedef vector<int> VI;  typedef vector<string> VS;  typedef vector<double> VD;  typedef long long LL;  typedef pair<int,int> PII;  int main() {  return 0;  } |