目录

[二、分数规划问题 1](#_Toc179673650)

[三、多项式相关 1](#_Toc179673651)

[四、集合卷积fwt && SOS dp 2](#_Toc179673654)

[五、字符串相关 2](#_Toc179673657)

[六、积性函数与筛法 5](#_Toc179673663)

[七、网络流 && 匹配问题 7](#_Toc179673666)

[八、计数问题&反演容斥&组合数学 8](#_Toc179673671)

[九、数论 9](#_Toc179673679)

[十、数据结构 10](#_Toc179673687)

[十一、图论问题 13](#_Toc179673694)

[十二、dp相关 17](#_Toc179673704)

[十三、树上问题 17](#_Toc179673711)

[十四、分治、分块、莫队 18](#_Toc179673716)

[十五、高斯消元 19](#_Toc179673721)

[十六、二进制技巧 19](#_Toc179673722)

[十七、计算几何常用函数 19](#_Toc179673723)

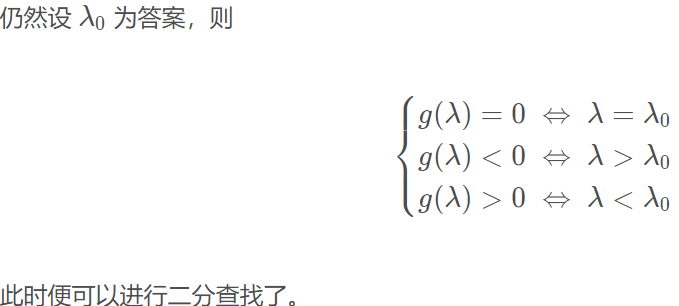
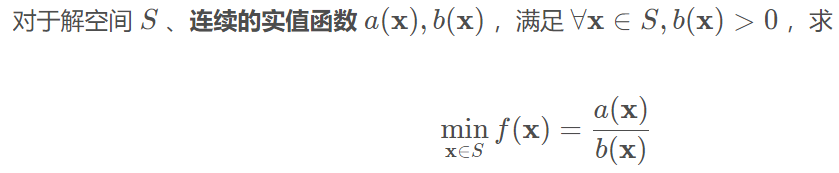
[多面体欧拉定理 25](#_Toc179673724)

[二十、代码查错 25](#_Toc179673725)

[二十一、Python Template 25](#_Toc179673726)

# 二、分数规划问题

**分数规划问题：（平均值、中位数等问题）**



*g*(*λ*)=min​[*a*(**x**)−*λ*⋅*b*(**x**)]

# 三、多项式相关

多项式模板

1. **namespace Poly{**
2. **ll upd(ll x) { return x + (x >> 63 & mod); }**
4. **#define VLL vector<ll>**
5. **int limit;**
6. **ll omg[N];**
8. **void pre\_ntt(int len) {**
9. **for (limit = 1; limit < len; limit <<= 1) ;**
10. **static int L = 1;**
11. **for (int &i = L; i < limit; i <<= 1) {**
12. **omg[i] = 1; int w = qpow(3, mod / 2 / i);**
13. **for (int j = 1; j < i; j++) omg[i + j] = omg[i + j - 1] \* w % mod;**
14. **}**
15. **}**
16. **void dft(VLL &p) {**
17. **for (int i = limit >> 1, s = limit; i; i >>= 1, s >>= 1)**
18. **for (int j = 0; j < limit; j += s) for (int k = 0, o = i; k < i; ++k, ++o) {**
19. **int x = p[j + k], y = p[i + j + k];**
20. **p[j + k] = upd(x + y - mod), p[i + j + k] = omg[o] \* upd(x - y) % mod;**
21. **}**
22. **}**
23. **void idft(VLL &p) {**
24. **for (int i = 1, s = 2; i < limit; i <<= 1, s <<= 1)**
25. **for (int j = 0; j < limit; j += s) for (int k = 0, o = i; k < i; ++k, ++o) {**
26. **int x = p[j + k], y = omg[o] \* p[i + j + k] % mod;**
27. **p[j + k] = upd(x + y - mod), p[i + j + k] = upd(x - y);**
28. **}**
29. **reverse(p.begin() + 1, p.end());**
30. **for (int i = 0, inv = qpow(limit, mod - 2); i < limit; i++) p[i] = p[i] \* inv % mod;**
31. **}**
32. **void ntt(VLL &p, int op) {**
33. **p.resize(limit);**
34. **if (op == 1) dft(p);**
35. **else idft(p);**
36. **}**
37. **VLL operator \* (VLL a, VLL b) {**
38. **int len = a.size() + b.size();**
39. **pre\_ntt(len);**
40. **ntt(a, 1), ntt(b, 1);**
41. **for (int i = 0; i < limit; i++) a[i] = a[i] \* b[i] % mod;**
42. **ntt(a, 0), a.resize(len);**
43. **return a;**
44. **}**
45. **VLL operator + (VLL a, const VLL &b) {**
46. **a.resize(max(a.size(), b.size()));**
47. **for (int i = 0; i < a.size(); i++)**
48. **a[i] = upd(a[i] + (i < b.size() ? b[i] : 0) - mod);**
49. **return a;**
50. **}**
51. **VLL operator - (VLL a, const VLL &b) {**
52. **a.resize(max(a.size(), b.size()));**
53. **for (int i = 0; i < a.size(); i++)**
54. **a[i] = upd(a[i] - (i < b.size() ? b[i] : 0));**
55. **return a;**
56. **}**
57. **VLL inv(VLL a, int n = -1) {**
58. **VLL res(1), t, t2;**
59. **assert(a[0]);**
60. **res[0] = qpow(a[0], mod - 2);**
61. **for (int l = 1; l < a.size(); l <<= 1) {**
62. **t2 = a, t2.resize(l << 1);**
63. **t = t2 \* res, t.resize(l << 1), t = t \* res, t.resize(l << 1);**
64. **res = res + res - t;**
65. **}**
66. **return res.resize(a.size()), res;**
67. **}**
68. **VLL diff(VLL a, int n) {**
69. **a.resize(n - 1);**
70. **for (int i = 0; i < n - 1; i++) a[i] = a[i + 1] \* (i + 1) % mod;**
71. **return a;**
72. **}**
73. **VLL integ(VLL a, int n) {**
74. **a.resize(n + 1);**
75. **for (int i = n; i; i--) a[i] = a[i - 1] \* ifac[i] % mod \* fac[i - 1] % mod; // inv[i] =  (ifac[i] \* fac[i - 1])**
76. **return a[0] = 0, a;**
77. **}**
78. **VLL ln(VLL a, int n) {**
79. **VLL b = inv(a, n);**
80. **a = diff(a, n), pre\_ntt(n << 1);**
81. **ntt(a, 1), ntt(b, 1);**
82. **for (int i = 0; i < limit; i++) a[i] = a[i] \* b[i] % mod;**
83. **ntt(a, 0), a.resize(n - 1);**
84. **return integ(a, n - 1);**
85. **}**
86. **VLL exp(VLL a, int n) {**
87. **if (n == 1) return {1};**
88. **a.resize(n); VLL b = exp(a, (n + 1) >> 1), c;**
89. **c = ln(b, n + 1), b.resize(n);**
90. **pre\_ntt(n + 1);**
91. **for (int i = 0; i < n; i++) a[i] = upd(a[i] - c[i] + (!i));**
92. **c = b, ntt(a, 1), ntt(c, 1);**
93. **for (int i = 0; i < limit; i++) a[i] = a[i] \* c[i] % mod;**
94. **ntt(a, 0), a.resize(n);**
95. **for (int i = (n + 1) >> 1; i < n; i++) b[i] = a[i];**
96. **return b;**
97. **}**
98. **}**

任意模数mtt：

1. **const int N = 4e6 + 100;**
2. **const int M = 32767;**
3. **const db pi = acos(-1);**
4. **const ll mod = 998244352;**
5. **struct cp{**
6. **db r, i;**
7. **cp(double r = 0, double i = 0) : r(r), i(i){}**
8. **cp operator \* (const cp &a) {return cp(r \* a.r - i \* a.i, r \* a.i + i \* a.r);}**
9. **cp operator + (const cp &a) {return cp(r + a.r, i + a.i);}**
10. **cp operator - (const cp &a) {return cp(r - a.r, i - a.i);}**
11. **}w[N], A[N], B[N], AA[N], BB[N];**
12. **int len, cc, wh[N];**
13. **cp conj(cp a)**
14. **{return cp(a.r, -a.i);}**
15. **void fft(cp \*a, bool inv) {**
16. **cp tmp;**
17. **for(int i = 0; i < len; i++)**
18. **if(i < wh[i])swap(a[i], a[wh[i]]);**
19. **for(int l = 2; l <= len; l <<= 1) {**
20. **int mid = l >> 1;**
21. **for(int i = 0; i < len; i += l) {**
22. **for(int j = 0; j < mid; j++) {**
23. **tmp = a[i + j + mid] \* (inv ? w[len - len / l \* j] : w[len / l \* j]);**
24. **a[i + j + mid] = a[i + j] - tmp;**
25. **a[i + j] = a[i + j] + tmp;**
26. **}**
27. **}**
28. **}**
29. **}**
30. **VLL mul(VLL &a, VLL &b) { // mtt with M = 32767**
31. **VLL res;**
32. **len = 1, cc = 0;**
33. **while(len < a.size() + b.size())**
34. **len <<= 1, ++cc;**
35. **for(int i = 1; i <= len; i++)**
36. **wh[i] = (wh[i >> 1] >> 1) | ((i & 1) << (cc - 1));**
37. **for(int i = 0; i <= len; i++)**
38. **w[i] = cp(cos(2.0 \* pi \* i / len), sin(2.0 \* pi \* i / len));**
39. **int sz = a.size() + b.size() - 1;**
40. **a.resize(len), b.resize(len);**
41. **for(int i = 0; i < len; i++) {**
42. **A[i] = cp(a[i] / M, a[i] % M);**
43. **B[i] = cp(b[i] / M, b[i] % M);**
44. **}**
45. **fft(A, 0), fft(B, 0);**
46. **for(int i = 0; i < len; i++) {**
47. **cp aa, bb, cc, dd;**
48. **int j = (len - i) % len;**
49. **aa = (A[i] + conj(A[j])) \* cp(0.5, 0);**
50. **bb = (A[i] - conj(A[j])) \* cp(0, -0.5);**
51. **cc = (B[i] + conj(B[j])) \* cp(0.5, 0);**
52. **dd = (B[i] - conj(B[j])) \* cp(0, -0.5);**
53. **AA[i] = aa \* cc + aa \* dd \* cp(0, 1);**
54. **BB[i] = bb \* dd + bb \* cc \* cp(0, 1);**
55. **}**
56. **fft(AA, 1), fft(BB, 1);**
57. **res.resize(sz);**
58. **for(int i = 0; i < sz; i++) {**
59. **ll ac, ad, bc, bd;**
60. **ac = (ll)(AA[i].r / len + 0.5) % mod;**
61. **ad = (ll)(AA[i].i / len + 0.5) % mod;**
62. **bd = (ll)(BB[i].r / len + 0.5) % mod;**
63. **bc = (ll)(BB[i].i / len + 0.5) % mod;**
64. **res[i] = (ac \* M \* M + (ad + bc) \* M + bd) % mod;**
65. **}**
66. **return res;**
67. **}**

另类分治fft：

求f（x）=sigma f（i）×f（x-i）

1. **if(l==r)  {**
2. **if(l==1)f[l]=1;**
3. **else Mul(f[l],jc[l-2]);**
4. **return;**
5. **}**
6. **int mid=(l+r)>>1;**
7. **sol(l,mid);**
8. **pre\_ntt(r-l+2);**
9. **for(int i=0;i<len;i++)A[i]=B[i]=0;**
10. **for(int i=l;i<=mid;i++)A[i-l]=F(i);//1LL\*(jc[i-1]-f[i]+mod)%mod\*ijc[i-1]%mod;**
11. **for(int i=1;i<=r-l;i++)B[i]=G(i);**
12. **ntt(A,0),ntt(B,0);**
13. **for(int i=0;i<len;i++)A[i]=1LL\*A[i]\*B[i]%mod;**
14. **ntt(A,1);**
15. **for(int i=mid+1;i<=r;i++)Ad(f[i],A[i-l]);**
16. **if(l>1)    {**
17. **for(int i=0;i<len;i++)A[i]=B[i]=0;**
18. **for(int i=l;i<=mid;i++)A[i-l]=G(i);//1LL\*(jc[i-1]-f[i]+mod)%mod\*ijc[i-1]%mod;**
19. **for(int i=1;i<=r-l;i++)B[i]=F(i);**
20. **ntt(A,0),ntt(B,0);**
21. **for(int i=0;i<len;i++)A[i]=1LL\*A[i]\*B[i]%mod;**
22. **ntt(A,1);**
23. **for(int i=mid+1;i<=r;i++)Ad(f[i],A[i-l]);**
24. **}**
25. **sol(mid+1,r);**

**一些应用**

1.fft做字符串匹配：一般用来搞例如通配符之类奇奇怪怪的匹配，构造F[i] =sigma (S[i] – T[i])^2，反转T串，再把平方拆开很容易发现是fft的形式。如果S中有通配符，就令通配符的值为0，构造F[i] = sigma(S[i] – T[i])^2 \* S[i]，然后类似的反转拆开，fft算。（warning：用ntt时给每个字符分配一个rand权值，不要只用ascii码，容易被卡！）

2.fft做可行性dp，保证所有物品价值的和大小在1e5以内时，先对物品价值进行排序，然后对于价值 vi物品构建生成函数1 + x^vi，分治fft相乘即可，复杂度nlog^2

# 四、集合卷积fwt && SOS dp

集合卷积：

And正变换相当于将每个位置的值加到这个位置所有的子集位置上去，逆变换相当于每个位置减到（下传减法）这个位置的所有子集位置上去。Or正变换相当于将每个位置的值加到这个位置的超集位置上去，逆变换相当于将每个位置的值减到这个位置的超集位置上去。Xor: F(i) = Sum of A[j], (j&i is odd in binary) – Sum of A[j], (j&i) is even in binary). 逆变换的唯一区别在于最后除以len。

1. **void And(ll \*a,bool inv)  {**
2. **for(int l=2,md=1;l<=len;l<<=1,md<<=1)**
3. **for(int i=0;i<len;i+=l)**
4. **for(int j=0;j<md;j++)**
5. **inv?Dw(a[i+j],a[i+j+md]):Ad(a[i+j],a[i+j+md]);**
6. **}**
7. **void Or(ll \*a,bool inv)  {**
8. **for(int l=2,md=1;l<=len;l<<=1,md<<=1)**
9. **for(int i=0;i<len;i+=l)**
10. **for(int j=0;j<md;j++)**
11. **inv?Dw(a[i+j+md],a[i+j]):Ad(a[i+j+md],a[i+j]);**
12. **}**
13. **void Xor(ll \*a,bool inv)  {**
14. **ll tp;**
15. **for(int l=2,md=1;l<=len;l<<=1,md<<=1)**
16. **for(int i=0;i<len;i+=l)**
17. **for(int j=0;j<md;j++)**
18. **{**
19. **tp=a[i+j+md];**
20. **a[i+j+md]=(a[i+j]-tp+mod)%mod;**
21. **Ad(a[i+j],tp);**
22. **if(inv)Mul(a[i+j],inv2),Mul(a[i+j+md],inv2);**
23. **}**
24. **}**

SOS dp *F*[*mask*]=*i*∈*mask*∑​*A*[*i*]

可以是一些十进制数，对每个数求所有数位都比它小的数的个数。

初始化dp数组，dp(val) = 数字val出现次数

则首先从低位到高位枚举数位i，然后从小到大枚举val，若val这一位是0，则dp值不变，否则dp(val)加上把val的第i位改小1的dp值，是一个高维前缀和的过程。

1. **for(int i = 0, bas = 1; i < 6; i++, bas \*= 10)  {**
2. **for(int s = 0; s <= lim; s++)  {**
3. **if((s / bas) % 10 < 9)**
4. **dp[s + bas] += dp[s];**
5. **}**
6. **}**

# 五、字符串相关

3.Z函数（exkmp）：

Z函数也称为扩展kmp，假设现在有一个串s和t，通过一个对t的O（n）时间预处理，之后可以线性时间对串s的每一个下标i，求以s[i]为开头能和串t匹配的最大长度。

具体做法是对t的每一个下标i求其能与t开头匹配的最大长度，求的时候利用一下之前已经求出的信息，匹配的时候同理。

1. **void sol()  {**
2. **string s, t;**
3. **cin >> t >> s;**
4. **int n, m;**
5. **n = s.length();**
6. **m = t.length();**
7. **VI Z(n, 0);**
8. **int lp, rp;**
9. **lp = rp = 0;**
10. **for(int i = 1; i < n; i++)   {**
11. **if(i <= rp && i + Z[i - lp] - 1 < rp)**
12. **Z[i] = Z[i - lp];**
13. **else  {**
14. **int k = max(0, rp - i + 1);**
15. **while(i + k < n && s[k] == s[i + k])**
16. **++k;**
17. **Z[i] = k;**
18. **}**
19. **if(i + Z[i] - 1 > rp)**
20. **lp = i, rp = i + Z[i] - 1;**
21. **}**
22. **Z[0] = n;**
23. **ll res = 0;**
24. **for(int i = 0; i < n; i++)**
25. **res ^= 1LL \* (i + 1) \* (Z[i] + 1);**
26. **cout << res << '\n';**
27. **res = 0;**
28. **lp = rp = -1;**
29. **VI ans(m, 0);**
30. **for(int i = 0; i < m; i++)   {**
31. **if(i <= rp && i + Z[i - lp] - 1 < rp)**
32. **ans[i] = Z[i - lp];**
33. **else  {**
34. **int k = max(0, rp - i + 1);**
35. **while(i + k < m && k < n && s[k] == t[i + k])**
36. **++k;**
37. **ans[i] = k;**
38. **}**
39. **if(i + ans[i] - 1 > rp)**
40. **lp = i, rp = i + ans[i] - 1;**
41. **res ^= 1LL \* (i + 1) \* (ans[i] + 1);**
42. **}**
43. **cout << res << '\n';**
44. **}**

4.ac自动机

约等于kmp进阶版， 可求多个串在其他多个串哪些位置出现。对于所有的查询串先建一棵trie树，然后类似kmp的失配数组那样求出每个节点失配指针，就是字典树里能和当前这个位置的后缀匹配最长的前缀（和kmp类似，当然也不能是它本身），求法就是对于节点x，沿着父亲的失配指针跳，直到跳到某个节点，它有和x节点代表字符相同的字符的孩子（或者是一直跳到根也没有），匹配也就类似kmp类比一下了。

一般对出现多个的串t1,t2, …,ti统一建一个ac自动机，用较长的某一串s在上面跑，可以处理诸如s的某一前缀的后缀，匹配ti的一些前缀这样的问题。

1. **struct Aho {**
2. **int nxt[N][26], fail[N], tot = 1, ed[N], vis[N], id\_mx;**
3. **vector<int> adj[N];**
4. **void init() {**
5. **for (int i = 0; i <= tot; i++) {**
6. **memset(nxt[i], 0, sizeof nxt[i]);**
7. **fail[i] = 0;**
8. **ed[i] = 0;**
9. **adj[i].clear();**
10. **vis[i] = 0;**
11. **}**
12. **tot = 1;**
13. **id\_mx = 0;**
14. **}**
15. **vector<int> ins(string s, int id) {**
16. **int n = s.length(), ps = 0, nw;**
17. **vector<int> cur;**
18. **cur.pb(ps);**
19. **id\_mx = max(id\_mx, id);**
20. **for (int i = 0; i < n; i++) {**
21. **nw = s[i] - 'a';**
22. **if (!nxt[ps][nw]) nxt[ps][nw] = tot++;**
23. **ps = nxt[ps][nw];**
24. **cur.pb(ps);**
25. **}**
26. **ed[ps] = id;**
27. **return cur;**
28. **}**
29. **void build() {**
30. **int nw, nx, tmp;**
31. **fail[0] = 0;**
32. **vector<int> que;**
33. **que.pb(0);**
34. **for (int q = 0; q < que.size(); q++) {**
35. **nw = que[q];**
36. **for (int i = 0; i < 26; i++) {**
37. **if (nxt[nw][i]) {**
38. **nx = nxt[nw][i];**
39. **if (!nw) fail[nx] = 0;**
40. **else {**
41. **tmp = fail[nw];**
42. **while (tmp && !nxt[tmp][i]) tmp = fail[tmp];**
43. **fail[nx] = nxt[tmp][i];**
44. **}**
45. **que.pb(nx);**
46. **}**
47. **}**
48. **}**
49. **for (int i = 1; i < tot; i++) {**
50. **adj[fail[i]].pb(i);**
51. **}**
52. **}**
53. **void match(string s) {**
54. **int n = s.length(), ps = 0;**
55. **for (int i = 0; i < n; i++) {**
56. **int nw = s[i] - 'a';**
57. **while (ps && !nxt[ps][nw]) ps = fail[ps];**
58. **ps = nxt[ps][nw];**
59. **++vis[ps];**
60. **}**
61. **}**
62. **void sol() {**
63. **}**
64. **}aho;**
65. **// usage: aho.init() aho.ins(), aho.build()**

5.后缀自动机：后缀自动机维护的每个节点，代表结束位置集合相同的多个子串，即前缀的后缀。做一些题时，可以反过来，变成后缀的前缀，此时fail树即后缀树，但是要处理出fail边代表的字符。

1. **const int SAM\_SZ = 2e6 + 100;**
2. **struct SAM {**
3. **bool ocr[SAM\_SZ][2]; // 多串改map**
4. **int tot, las, nxt[SAM\_SZ][26], fa[SAM\_SZ], mx[SAM\_SZ], mx\_ed[SAM\_SZ];**
5. **vector<int> adj[SAM\_SZ]; // fail tree (parent tree), edge sorted in lexi order for suffix tree**
6. **void init() {**
7. **for (int i = 0; i <= tot; i++) {**
8. **memset(nxt[i], 0, sizeof nxt[i]);**
9. **fa[i] = mx[i] = 0;**
10. **ocr[i][0] = ocr[i][1] = 0;**
11. **mx\_ed[i] = -1;**
12. **adj[i].clear();**
13. **}**
14. **tot = 1, las = 1;**
15. **}**
16. **int mn(int p) {**
17. **return mx[fa[p]] + 1;**
18. **}**
19. **void ins(int x, int op = 0, int ps = -1) {**
20. **int p = las, np = nxt[las][x], t, nt;**
21. **if (np) { // 广义用**
22. **if (mx[p] + 1 == mx[np]) {**
23. **las = np;**
24. **ocr[np][op] = 1;**
25. **} else {**
26. **nt = ++tot;**
27. **mx[nt] = mx[p] + 1;**
28. **memcpy(nxt[nt], nxt[np], sizeof nxt[np]);**
29. **fa[nt] = fa[np], fa[np] = nt;**
30. **while (p && nxt[p][x] == np) {**
31. **nxt[p][x] = nt, p = fa[p];**
32. **}**
33. **ocr[nt][op] = 1;**
34. **las = nt;**
35. **}**
36. **return;**
37. **}**
38. **np = ++tot, mx[np] = mx[p] + 1;**
39. **las = np, ocr[np][op] = 1, mx\_ed[np] = ps;**
40. **while (p && !nxt[p][x]) {**
41. **nxt[p][x] = np;**
42. **p = fa[p];**
43. **}**
44. **if (!p) {**
45. **fa[np] = 1;**
46. **return;**
47. **}**
48. **t = nxt[p][x];**
49. **if (mx[t] == mx[p] + 1) {**
50. **fa[np] = t;**
51. **} else {**
52. **nt = ++tot, mx[nt] = mx[p] + 1;**
53. **memcpy(nxt[nt], nxt[t], sizeof nxt[t]);**
54. **fa[nt] = fa[t], fa[t] = fa[np] = nt;**
55. **while (p && nxt[p][x] == t) {**
56. **nxt[p][x] = nt;**
57. **p = fa[p];**
58. **}**
59. **}**
60. **}**
61. **void build\_edge(string &s) { // 出边满足后缀树字典序，需要插入反串**
62. **for (int i = 2; i <= tot; i++) {**
63. **adj[fa[i]].pb(i);**
64. **}**
65. **auto dfs = [&](auto self, int x) -> void{**
66. **for (int y : adj[x]) {**
67. **self(self, y);**
68. **mx\_ed[x] = max(mx\_ed[x], mx\_ed[y]);**
69. **}**
70. **};**
71. **dfs(dfs, 1);**
72. **for (int i = 1; i <= tot; i++) {**
73. **sort(all(adj[i]), [&](int x, int y) {**
74. **int cx = s[mx\_ed[x] - mx[i]];**
75. **int cy = s[mx\_ed[y] - mx[i]];**
76. **return cx < cy;**
77. **});**
78. **}**
79. **}**
80. **void build(string &s) {**
81. **init();**
82. **for (int i = 0; i < s.length(); i++) {**
83. **ins(s[i] - 'a', 0, i);**
84. **}**
85. **build\_edge(s);**
86. **}**
87. **vector<int> match(string t) {**
88. **vector<int> cur;**
89. **int nw = 1, cur\_len = 0;**
90. **cur.pb(nw);**
91. **for (int i = 0; i < t.length(); i++) {**
92. **int x = t[i] - 'a';**
93. **while (nw > 1 && !nxt[nw][x]) {**
94. **nw = fa[nw];**
95. **cur\_len = mx[nw];**
96. **}**
97. **if (nxt[nw][x]) nw = nxt[nw][x], cur\_len++;**
98. **cur.pb(nw);**
99. **}**
100. **return cur;**
101. **}**
102. **void sol(int n) {**
103. **}**
104. **}sam;**

6.后缀数组//rnk[i]表示下标i起始后缀排名是多少，sa[i]表示排名第i小的起始位置下标，ht[i] = lcp(sa[i], sa[i – 1])

1. **const int N = 2e6 + 100; // 2 \* string\_length**
2. **struct SA {**
3. **int n;**
4. **string s;**
5. **int m, x[N], y[N], bin[N], rnk[N], sa[N], ht[N];**
6. **// m number of dif vals, x[i] = suf i's first dimension val**
7. **// y[i] = who is rank i in second dimension**
8. **void rsort() {**
9. **for (int i = 0; i <= m; i++) bin[i] = 0;**
10. **for (int i = 1; i <= n; i++) bin[x[i]]++;**
11. **for (int i = 1; i <= m; i++) bin[i] += bin[i-1];**
12. **for (int i = n; i >= 1; i--) sa[bin[x[y[i]]]--] = y[i];**
13. **}**
14. **void cal(int \_n, string \_s)**
15. **{**
16. **n = \_n, s = \_s;**
17. **int num;**
18. **for (int i = 0; i <= 2 \* n; i++) { // remember clear!!**
19. **x[i] = y[i] = bin[i] = rnk[i] = sa[i] = ht[i] = 0;**
20. **}**
21. **for(int i = 1; i <= n; i++){**
22. **x[i] = s[i], y[i] = i;**
23. **}**
24. **m = 130, rsort();**
25. **for (int l = 1; l <= n; l <<= 1) {**
26. **num = 0;**
27. **for (int i = n - l + 1; i <= n; i++) y[++num] = i;**
28. **for (int i = 1; i <= n; i++) {**
29. **if(sa[i] > l) {**
30. **y[++num] = sa[i] - l;**
31. **}**
32. **}**
33. **rsort();**
34. **for (int i = 0; i <= n; i++) {**
35. **swap(x[i], y[i]);**
36. **}**
37. **num = x[sa[1]] = 1;**
38. **for(int i = 2; i <= n; i++) {**
39. **x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + l] == y[sa[i - 1] + l])? num: ++num;**
40. **}**
41. **if(num == n)break;**
42. **m = num;**
43. **}**
44. **for (int i = 1; i <= n; i++) {**
45. **rnk[sa[i]] = i;**
46. **}**
47. **int k = 0;**
48. **for (int i = 1, j; i <= n; i++) {**
49. **if(rnk[i] == 1) {**
50. **ht[1] = k = 0;**
51. **continue;**
52. **}**
53. **if (k) --k;**
54. **j = sa[rnk[i] - 1];**
55. **while (i + k <= n && j + k <= n && s[i + k] == s[j + k]) ++k;**
56. **ht[rnk[i]] = k;**
57. **}**
58. **}**
59. **} sa;**

后缀数组处理本质不同字串的话，可以考虑sa[i]为开头没有被之前统计的字串，发现之前被统计的数量就是h[i]，减掉即可。

后缀树，串反转建后缀自动机，parent树就是后缀树，不过每个边具体是什么需要额外维护。如果要看后缀树上表示的边到底压缩了哪些字母我们可以考虑，在后缀自动机上用我当前的节点能表示的最长串-fa[当前节点]表示的最长串，然后将其反过来就是后缀树上压缩的边了。

fft字符串匹配，通配之类问题，见多项式。

询问一些多个串匹配的最大长度，类似后缀数组的处理，将多个串排序，求lcp数组。

7.回文自动机，对奇数长度偶数长度各建树，奇根表示长度-1，偶根表示长度0，转移边表示同时xx跳，fail直接暴力找。

1. **const int N = 1e6 + 100;**
2. **struct PAM{**
3. **int tot, len[N], las, nxt[N][26], dis[N], fail[N], ocr[N];**
4. **int dep[N];**
5. **vector<int> ed[N];**
6. **string s;**
7. **int n;**
8. **void ins(int c, int ps)    {**
9. **int nw = las;**
10. **while(ps - len[nw] - 1 < 0 || c != s[ps - len[nw] - 1] - 'a')  {**
11. **nw = fail[nw];**
12. **}**
13. **if(nxt[nw][c]) {**
14. **las = nxt[nw][c];**
15. **ed[las].pb(ps);**
16. **ocr[las]++;**
17. **return;**
18. **}**
19. **int np = ++tot;**
20. **len[np] = len[nw] + 2;**
21. **int t = fail[nw];**
22. **while(ps - len[t] - 1 < 0 || c != s[ps - len[t] - 1] - 'a') {**
23. **t = fail[t];**
24. **}**
25. **fail[np] = nxt[t][c];**
26. **nxt[nw][c] = np;**
27. **las = np;**
28. **ed[np].pb(ps);**
30. **dep[las] = dep[fail[las]] + 1;**
31. **ocr[las]++;**
32. **}**
33. **void init(string \_s, int \_n)    {**
34. **s = \_s, n = \_n;**
35. **tot = 1;**
36. **len[1] = -1;**
37. **len[0] = 0;**
38. **dis[0] = dis[1] = 0;**
39. **fail[0] = 1;**
40. **dep[0] = dep[1] = 0;**
41. **las = 0;**
42. **for (int i = 0; i < n; i++) {**
43. **ins(s[i] - 'a', i);**
44. **}**
45. **}**
46. **void sol() {**
47. **}**
48. **}pam;**

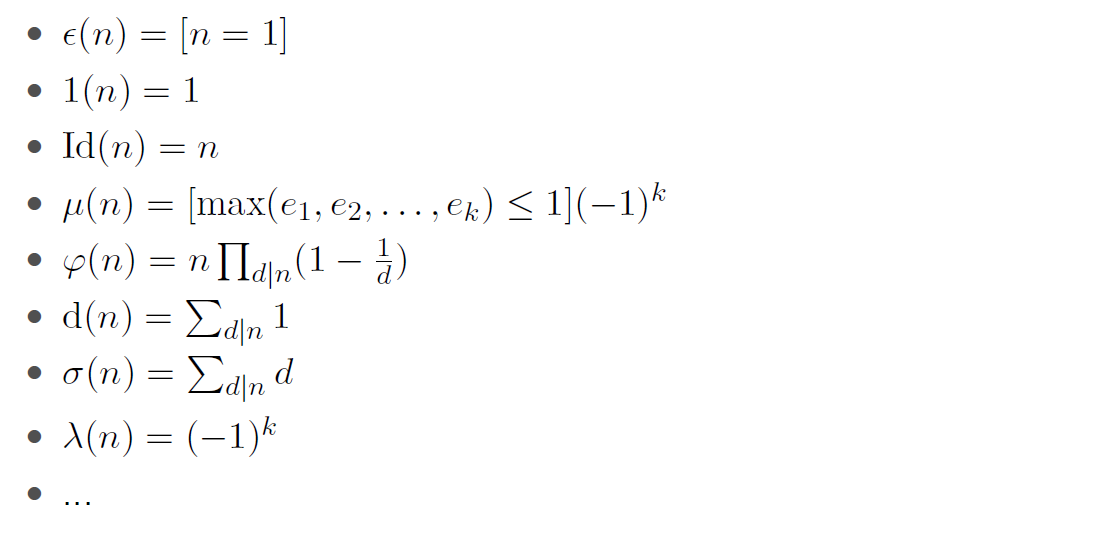
# 六、积性函数与筛法

积性函数即gcd(a, b) = 1时满足f(ab) = f(a) \* f(b)的函数

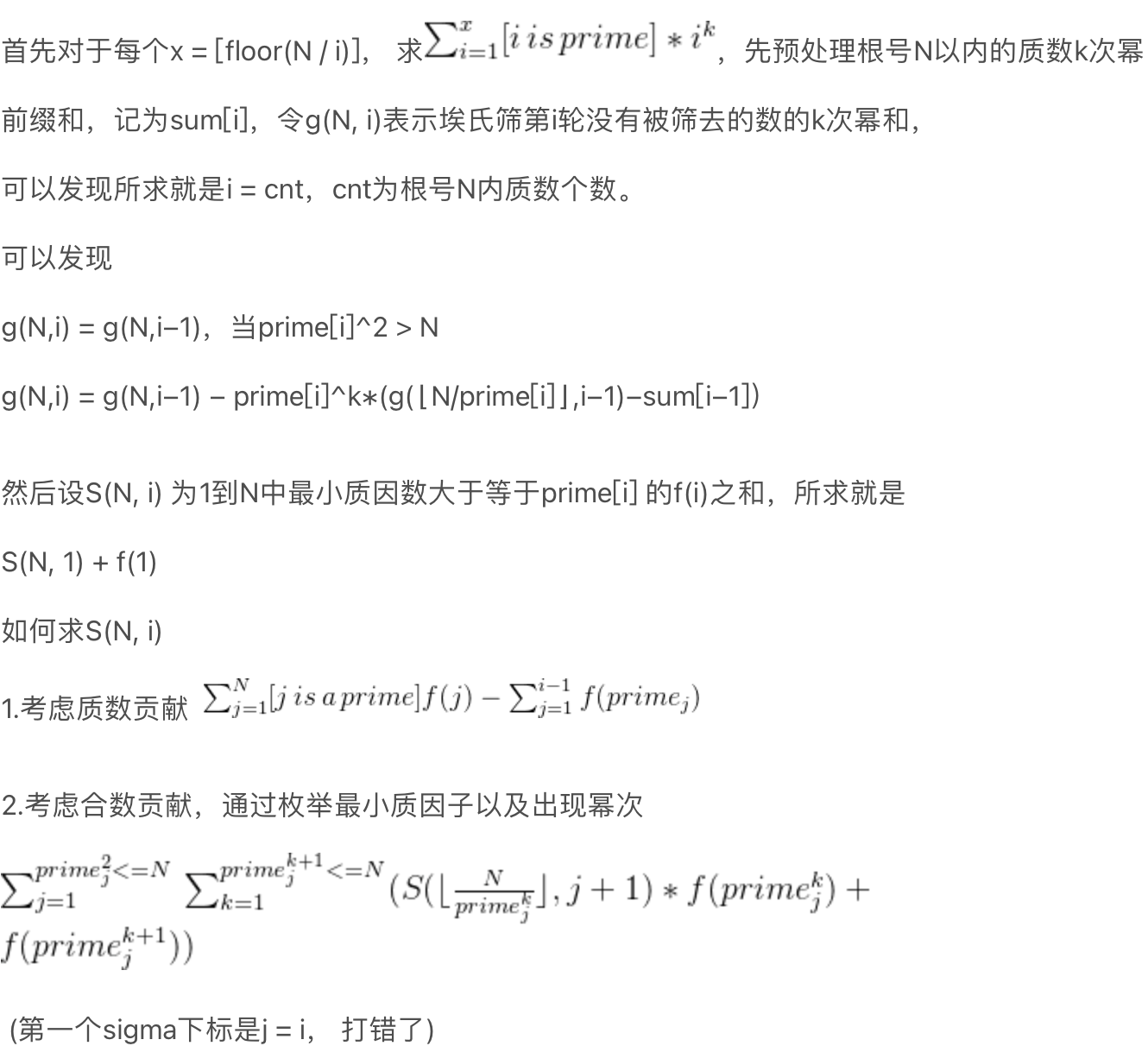
迪利克雷卷积：

f\*g(n) = sigma f(d) \* g(n / d) 其中d|n

常用积性函数及其迪利克雷卷积



μ∗1=ϵ【莫比乌斯反演】【μ与1互为逆元】ϕ∗1=Id ，ϕ=Id∗μ ，d=1∗1， 1=μ∗d



以上为min\_25筛做法，有时一些函数不是积性，但满足一些特殊性质，也可以用min25

1. **namespace siever {**
2. **const int N = 2e5 + 100;**
3. **ll n, B, num;**
4. **ll val[N], g[2][N], sum[2][N], f[N], pre[N];**
5. **int ps1[N], ps2[N];**
6. **int pos(ll x) {**
7. **if(x <= B)**
8. **return ps1[x];**
9. **return ps2[n / x];**
10. **}**
11. **int pnum;**
12. **int pri[N];**
13. **bool np[N];**
14. **void work(ll \_n) {**
15. **n = \_n;**
16. **B = sqrt(n);**
17. **/\***
18. **sum[k][i] = Prefix Sum of [pri\_i ^ k]**
19. **\*/**
20. **for (int i = 2; i <= B; i++) {**
21. **if(!np[i]) {**
22. **pri[++pnum] = i;**
23. **sum[0][pnum] = pnum;**
24. **sum[1][pnum] = (sum[1][pnum - 1] + i) % mod;**
25. **}**
26. **for(int j = 1; j <= pnum && i \* pri[j] <= B; j++) {**
27. **np[i \* pri[j]] = 1;**
28. **if(i % pri[j] == 0)**
29. **break;**
30. **}**
31. **}**
32. **/\***
33. **Give index to Each Block of [n / d]**
34. **Small Index -> Bigger Original Value**
36. **g[k][i] Indicates Prefix Sum of (i ^ k), Notice it Starts From Index 2nd**
37. **\*/**
38. **for (ll l = 1, r; l <= n; l = r + 1) {**
39. **ll x = n / l;**
40. **r = n / x;**
41. **val[++num] = x;**
42. **if(x <= B)**
43. **ps1[x] = num;**
44. **else ps2[n / x] = num;**
45. **x %= mod;**
46. **g[0][num] = (x - 1 + mod) % mod;**
47. **g[1][num] = (x - 1 + mod) % mod \* (x + 2) / 2 % mod;**
48. **}**
49. **/\***
50. **Calculate Prefix Sum of (i ^ k) with [i is Prime]**
51. **g[k][num] = Prefix Sum from (1 to val[num]) of (i ^ k) with [i is Prime]**
52. **\*/**
53. **for(int i = 1; i <= pnum; ++i) {**
54. **for(int j = 1; j <= num && 1LL \* pri[i] \* pri[i] <= val[j]; ++j) {**
55. **int bf = pos(val[j] / pri[i]);**
56. **g[0][j] = (g[0][j] - g[0][bf] + sum[0][i - 1] + mod) % mod;**
57. **g[1][j] = (g[1][j] - pri[i] \* (g[1][bf] - sum[1][i - 1] + mod) % mod + mod) % mod;**
58. **}**
59. **}**
60. **/\***
61. **Calculate the contribution of Prime Position to Final answer**
62. **f[i] = the contribution of Prime Position to Final answer in [1 to val[i]]**
64. **Calculate Pre[i] = Contribution of the [1st Prime to i'th Prime]**
65. **\*/**
66. **for(int i = 1; i <= num; i++)**
67. **f[i] = (g[1][i] - g[0][i] + mod) % mod;**
68. **for(int i = 1; i <= pnum; i++)**
69. **pre[i] = (sum[1][i] - sum[0][i] + mod) % mod;**
70. **for(int j = pnum; j >= 1; j--) {**
71. **for(int i = 1; i <= num; i++) {**
72. **if(1LL \* pri[j] \* pri[j] > val[i])**
73. **break;**
74. **ll tmp = pri[j];**
75. **for(int e = 1; tmp <= val[i] / pri[j]; e++, tmp \*= pri[j]) {**
76. **/\***
77. **s = Contribution of [Prime\_{j} ^ e]**
78. **t = Contribution of [Prime\_{j} ^ (e+1)]**
79. **\*/**
80. **ll s = (tmp - tmp / pri[j]) % mod;**
81. **ll t = (tmp \* pri[j] - tmp) % mod;**
82. **(f[i] += (s \* (f[pos(val[i] / tmp)] - pre[j] + mod) + t)) %= mod;**
83. **}**
84. **}**
85. **}**
86. **// 1 should be replaced by Contribution of index 1**
87. **cout << (f[1] + 1) % mod;**
88. **}**
89. **}**

最短线性递推式：（bm）

求一个r数组，, 且m最短。

1. **int cnt, fail[MAXN];**
2. **double val[MAXN], delta[MAXN];**
3. **vector <double> ans[MAXN];**
4. **int main() {**
5. **int n; read(n);**
6. **for (int i = 1; i <= n; i++)**
7. **scanf("%lf", &val[i]);**
8. **for (int i = 1; i <= n; i++) {**
9. **double tmp = val[i];**
10. **for (unsigned j = 0; j < ans[cnt].size(); j++)**
11. **tmp -= ans[cnt][j] \* val[i - j - 1];**
12. **delta[i] = tmp;**
13. **if (fabs(tmp) <= eps) continue;**
14. **fail[cnt] = i;**
15. **if (cnt == 0) {**
16. **ans[++cnt].resize(i);**
17. **continue;**
18. **}**
19. **double mul = delta[i] / delta[fail[cnt - 1]];**
20. **cnt++; ans[cnt].resize(i - fail[cnt - 2] - 1);**
21. **ans[cnt].push\_back(mul);**
22. **for (unsigned j = 0; j < ans[cnt - 2].size(); j++)**
23. **ans[cnt].push\_back(ans[cnt - 2][j] \* -mul);**
24. **if (ans[cnt].size() < ans[cnt - 1].size()) ans[cnt].resize(ans[cnt - 1].size());**
25. **for (unsigned j = 0; j < ans[cnt - 1].size(); j++)**
26. **ans[cnt][j] += ans[cnt - 1][j];**
27. **}**
28. **for (unsigned i = 0; i < ans[cnt].size(); i++)**
29. **cout << ans[cnt][i] << ' ';**
30. **return 0;**
31. **}**

# 七、网络流 && 匹配问题

最大流Dinic：

1. **const int N = 1e6 + 100;**
2. **struct Dinic {**
3. **int s, t;  // 汇点t要是最大编号点**
4. **int hed[N], cur[N], dep[N], nxt[N], las[N], to[N], cnt;**
5. **bool bfs() {**
6. **memset(dep, 0, (t + 5) \* sizeof (int));**
7. **dep[s] = 1;**
8. **vector<int> que;**
9. **que.pb(s);**
10. **for (int q = 0; q < que.size(); q++) {**
11. **int nw = que[q];**
12. **for (int i = hed[nw]; ~i; i = nxt[i]) {**
13. **if (!dep[to[i]] && las[i]) {**
14. **dep[to[i]] = dep[nw] + 1;**
15. **que.pb(to[i]);**
16. **}**
17. **}**
18. **}**
19. **return dep[t];**
20. **}**
21. **int dfs(int ps,int flow) {**
22. **if(ps == t) return flow;**
23. **for (int &i = cur[ps]; ~i; i = nxt[i]) {**
24. **if (dep[to[i]] == dep[ps] + 1 && las[i]) {**
25. **int nw = dfs(to[i], min(flow, las[i]));**
26. **if (nw) {**
27. **las[i] -= nw;**
28. **las[i ^ 1] += nw;**
29. **return nw;**
30. **}**
31. **}**
32. **}**
33. **return 0;**
34. **}**
35. **void add\_edge(int u,int v,int w)//加边**
36. **{**
37. **nxt[++cnt] = hed[u], las[cnt] = w, to[cnt] = v, hed[u] = cnt;**
38. **nxt[++cnt] = hed[v], las[cnt] = 0, to[cnt] = u, hed[v] = cnt;**
39. **}**
40. **void init(int \_s, int \_t)**
41. **{**
42. **memset(nxt,-1,sizeof nxt);**
43. **memset(hed,-1,sizeof hed);**
44. **cnt = -1;**
45. **s = \_s, t = \_t;**
46. **return;**
47. **}**
48. **ll max\_flow() {**
49. **ll flow = 0;**
50. **while (bfs()) {**
51. **for (int i = 0; i <= t; i++) cur[i] = hed[i];**
52. **ll tmp;**
53. **while ((tmp = dfs(s, (ll)1e18)) != 0) {**
54. **flow += tmp;**
55. **}**
56. **}**
57. **return flow;**
58. **}**
59. **}dinic;**

最小费用最大流mcmf：

1. **const int N = 1e6 + 100;**
2. **struct Mcmf {**
3. **int s, t;**
4. **bool inq[N];**
5. **int pre[N], dis[N], hed[N], nxt[N], las[N], to[N], cost[N], cnt;**
6. **bool spfa()**
7. **{**
8. **memset(inq, 0, (t + 5) \* sizeof (int));**
9. **memset(pre, -1, (t + 5) \* sizeof (int));**
10. **memset(dis, 127, (t + 5) \* sizeof (int));**
11. **vector<int> que;**
12. **dis[s] = 0, inq[s] = 1, que.pb(s);**
13. **for (int q = 0; q < que.size(); q++) {**
14. **int nw = que[q];**
15. **inq[nw] = 0;**
16. **for (int i = hed[nw]; ~i; i = nxt[i]) {**
17. **int v = to[i];**
18. **if (las[i] && dis[v] > dis[nw] + cost[i]) {**
19. **dis[v] = dis[nw] + cost[i];**
20. **pre[v] = i;**
21. **if (!inq[v]) {**
22. **que.pb(v);**
23. **inq[v] = 1;**
24. **}**
25. **}**
26. **}**
27. **}**
28. **return dis[t] <= (int)1e9;**
29. **}**
30. **pair<ll, ll> mcmf()**
31. **{**
32. **ll flow = 0, nw = 0, spd = 0;**
33. **while(spfa())**
34. **{**
35. **nw = 1e9;**
36. **for (int i = pre[t]; ~i; i = pre[to[i ^ 1]]) {**
37. **nw = min(nw, las[i]);**
38. **}**
39. **for (int i = pre[t]; ~i; i = pre[to[i ^ 1]]) {**
40. **las[i] -= nw;**
41. **las[i ^ 1] += nw;**
42. **}**
43. **flow += nw, spd += nw \* dis[t];**
44. **}**
45. **return make\_pair(flow, spd);**
46. **}**
47. **void init(int \_s, int \_t)**
48. **{**
49. **memset(nxt,-1,sizeof nxt);**
50. **memset(hed,-1,sizeof hed);**
51. **cnt = -1;**
52. **s = \_s, t = \_t;**
53. **return;**
54. **}**
55. **void add\_edge(int u,int v,int w, int c)//加边**
56. **{**
57. **nxt[++cnt] = hed[u], las[cnt] = w, to[cnt] = v, hed[u] = cnt, cost[cnt] = c;**
58. **nxt[++cnt] = hed[v], las[cnt] = 0, to[cnt] = u, hed[v] = cnt, cost[cnt] = -c;**
59. **}**
60. **};**

数据范围较大时，可能会有优秀的增广策略。

网络流建图技巧：

1.把有关系的点之类的建边,对于题目条件转化成对于流量或费用的的限制.

2.搞清楚是最大流还是费用流…

3.常用模型：

**（1）**点覆盖、最小点覆盖：点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点” 覆盖了所有“边”。**（2）**最小点覆盖：点最少的点覆盖。**（3）**点覆盖数：最小点覆盖的点数。**（4）**独立集：独立集即一个点集，集合中任两个结点不相邻，则称V为独立集。**（5）**最大独立集：点最多的独立集。**（6）**独立数：最大独立集的点。**（7）**若把上面最小点覆盖和最大独立集中的端点数改成点的权值，分别就是最小点权覆盖和最大点权独立集的定义。**（8）**最大点权独立集=总权值-最小点权覆盖集，最小点权覆盖集=图的最小割值=最大流。

**（9）**最大权闭合子图：选择某个东西可以得到一定收益，但选了它就必须选择它的一些后继，而不同物品可能有一些相同后继，求能选到的最大收益。做法：建源点s，向正权点连流量为点权的边，正权点再向它后继连流量inf的边，建汇点t，负权点向汇点连点权绝对值边，跑一波最小割（最大流），然后答案就是正权点和-最小割代价.**（10）**转化的一些技巧,如要求最大费用流可以把所有边费用取负后做,答案再取负,还有一些对于点的限制,可以把点一分为二,在拆出来的两个点间建边作为原来限制…

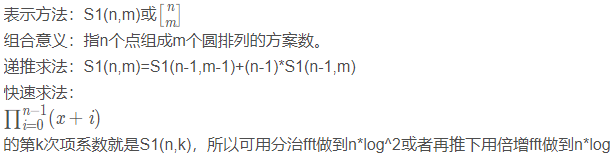
# 八、计数问题&反演容斥&组合数学

1.卡特兰数：统计n个元素出栈的方案数，h(n)=C(2n,n)/(n+1) =C(2n,n) - C(2n,n-1)

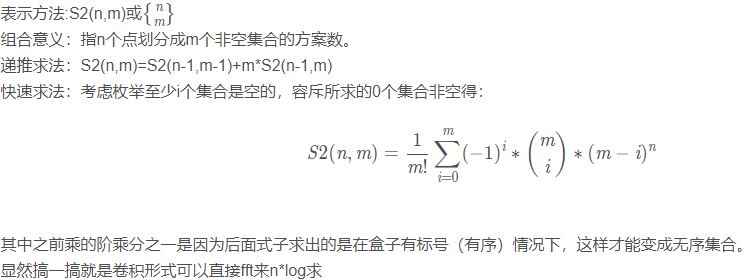
看成走折线图，每次执行一次进栈或出栈操作横坐标加1，同时若进栈纵坐标加1、出栈纵坐标-1，问题等价于从(0,0)到(2n,0)且不碰到y=-1直线方案数。假设某个方案碰到了-1，我们考察该方案从原点第一次碰到-1的走法，把每一步都取反(向上向下互换)，发现可以等价于从(0,-2)碰到-1，那么可能会碰到-1的方案数等价于从(0,-2)走到（2n,0）的方案数，即C(2n, n-1)，故最终方案数C(2n,n) - C(2n,n-1)

2.斯特林数：

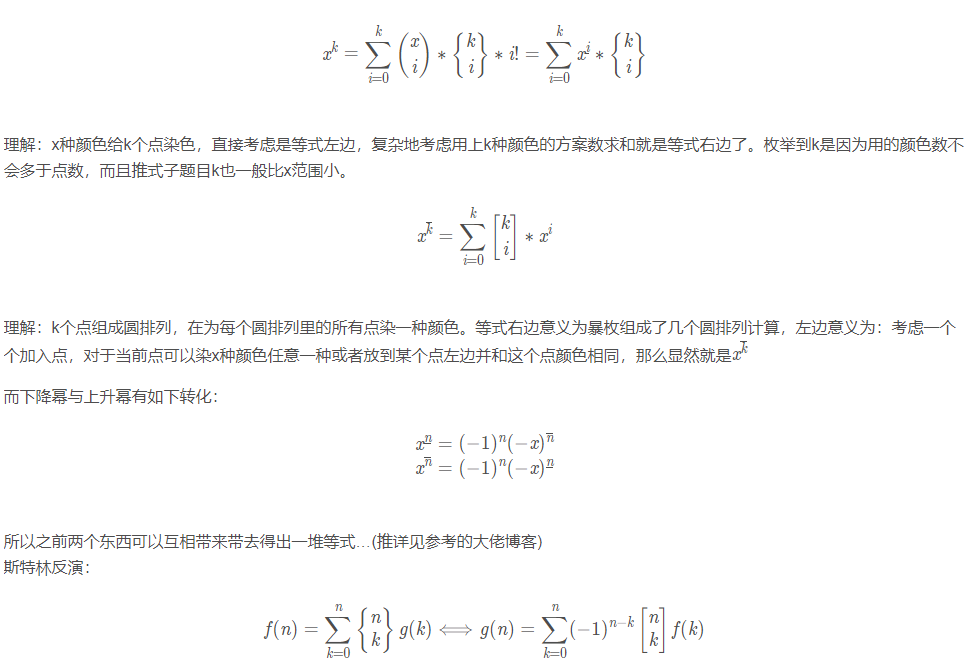
**（1）第一类：**



**（2）第二类：**

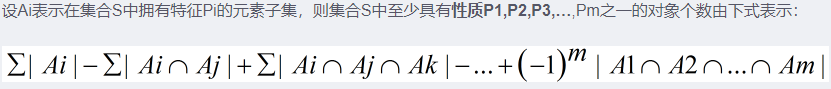


（4）常用斯特林数公式：

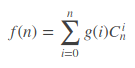


3.容斥：

**(1).最经典的排斥包含原理：**



**(2).二项式反演；**

 可以推出： 

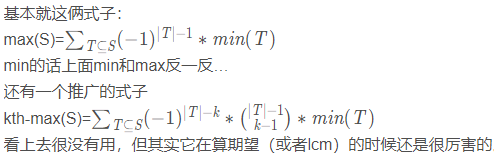
**(3).莫比乌斯反演：**

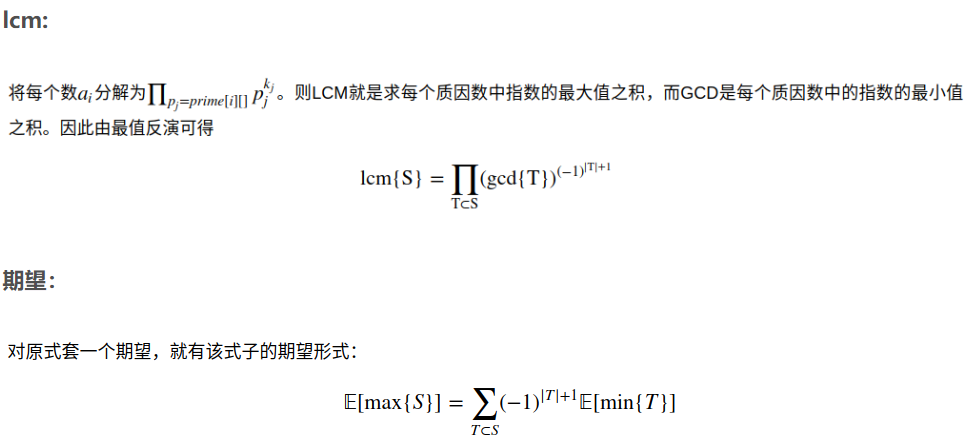
 可以推出：

**(4).斯特林反演：**

见之前斯特林数。

**(5).min-max容斥：**





min-max容斥经典问题，每次（随机）覆盖一个点，问某个集合内元素都被完全覆盖的期望时间 （即集合内所有点覆盖时间最大值期望）《=》 枚举集合中第一个点被覆盖的时间（最小值期望），套用min-max容斥。

比如树上一些点集被完全覆盖，之后还可以转化成（1-这些点划分连通块后只在连通块内取的概率）分之一，上树形dp（可能需要包括容斥系数-1的幂次）。

5.组合数前缀和

（1）第一种形式：把C(n,n)看成C(n + 1, n + 1),由组合数递推公式从左到右依次合并

C(n, n) + C(n + 1, n) + C(n + 2, n) + ... + C(m, n) = C(m + 1, n + 1)，（m >= n）

（2）第二种形式：

C(n, 0) + C(n, 1) + C(n, 2) + ... + C(n, m) 其中 m <= n

记所求 = S(n, m)

S(n, m + 1) = S(n, m) + C(n, m + 1)

S(n + 1, m) = 2 \* S(n, m) - C(n, m)

一些题目里m是恒定值, 然后需要代入不同的n求S(n, m)，这时候就需要第二个递推式了

如果询问n, m都是变量，可以暴力预处理f(x, y)其中x = k \* sqrt(n), y = k \* sqrt(m)，然后询问的时候找最接近的f(x, y)暴力走，复杂度是n \* sqrt(n)

6.组合数恒等式

(1) //左式枚举前n个被选中几个

(2) //左式枚举第i+1个球所在位置

7.同构问题，burnside引理



给定一置换集，定义两个方案相同为两方案可经过置换集中某一置换得到（例如环上一些点染色问题，存在n个置换，就是逆时针旋转1,2,3…n个）。本质不同的方案数 = 每个置换下不同点数 / 不同置换个数（即置换不动点均值）。（环上一些点问题，对于逆时针转x个置换，不动点就是逆时针转x个仍然和初始状态完全相同的染色方案）。

# 九、数论

1.质因数分解：

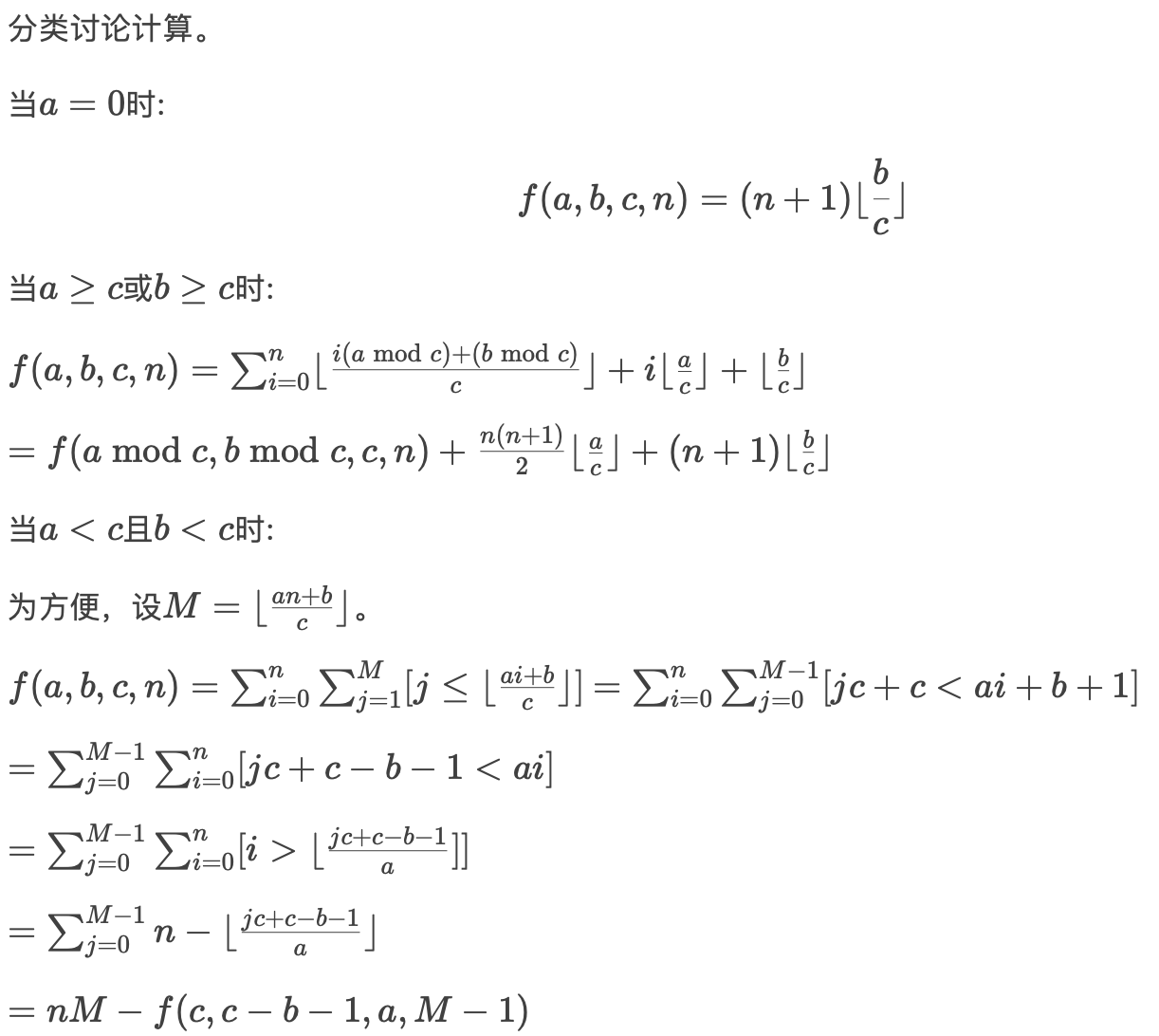
值域不大时，可以直接线性筛出每个数最小质因子（记录线性筛的时候每个数被谁筛的），然后每个值往下dfs即可分解。值域大时，暴力枚举素因子分解，dfs枚举因数：

2.扩展欧几里得

1. **int ex\_gcd(int a,int b,int &x,int &y)**
2. **{   int d=a;**
3. **if(b)**
4. **{**
5. **d=ex\_gcd(b,a%b,x,y);**
6. **x-=(a/b)\*y;swap(x,y);**
7. **}**
8. **else**
9. **x=1,y=0;**
10. **return d;**
11. **}**

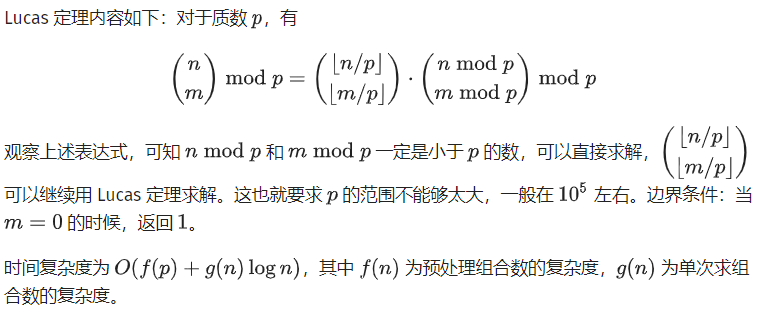
3类欧几里得算法

用于求：(要保证系数非负数，如果有负数，处理的时候要注意对负数的下取整，可能需要调整成对正数的上取整)



1. **ll floor\_sum(ll a, ll b, ll c, ll n) { // sum i=0...n, floor((a\*i + b)/c)**
2. **if (a == 0) {**
3. **return (n + 1) \* (b / c) % mod;**
4. **}**
5. **ll mx = (a \* n + b) / c; // 0, mx - 1**
6. **if (mx == 0) {**
7. **return 0;**
8. **}**
9. **if (a >= c || b >= c) {**
10. **return (n \* (n + 1) / 2 % mod \* (a / c) % mod + (n + 1) \* (b / c) % mod + floor\_sum(a % c, b % c, c, n)) % mod;**
11. **}**
12. **return ((mx \* (n + 1) - floor\_sum(c, a + c - b - 1, a, mx - 1)) % mod + mod) % mod;**
13. **}**

4.卢卡斯定理

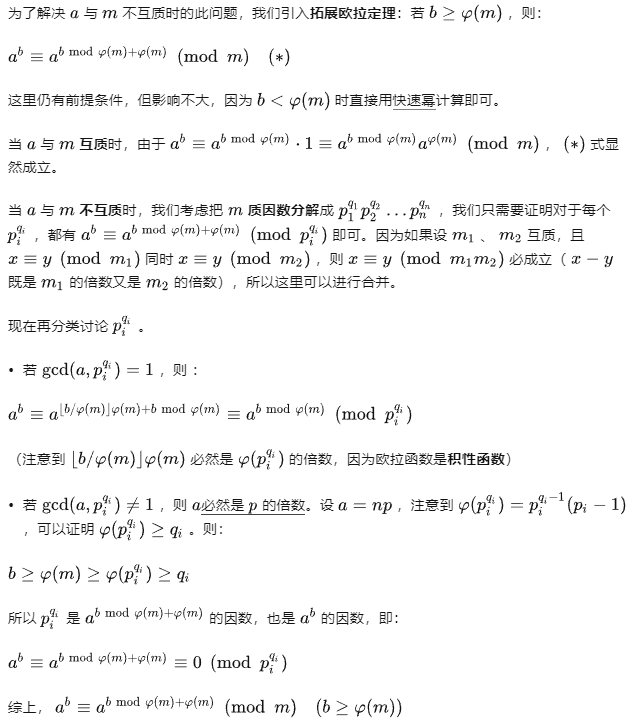


5.皮克定理：

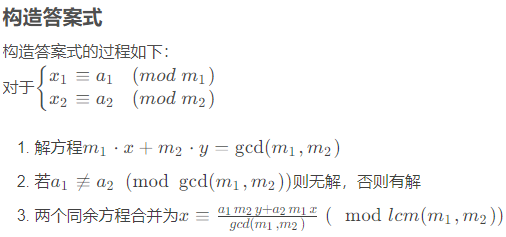
点阵中多边形（顶点在格点）面积**S=a+b/2-1**,其中**a**是严格在图形内部格点数，b是图形边界上格点数。应用，对面积和内部格点的要求可以转化为对边界格点要求，然后运用同余知识。

6.欧拉定理&扩展欧拉定理

欧拉定理：  
扩展欧拉定理：



7.扩展中国剩余定理excrt

****

# 十、数据结构

0.线段树

1. **template<class Info, class Lazy>**
2. **struct SegTree {**
3. **#define ls k << 1**
4. **#define rs k << 1 | 1**
5. **#define mid ((l + r) >> 1)**
6. **int n;**
7. **vector<Info> seg;**
8. **vector<bool> tag;**
9. **vector<Lazy> laz;**
11. **void push\_up(int k) {**
12. **seg[k] = seg[ls] + seg[rs];**
13. **}**
14. **void change(int k, Lazy val) {**
15. **seg[k] = seg[k] + val;**
16. **if (tag[k]) {**
17. **laz[k] = laz[k] + val;**
18. **} else {**
19. **laz[k] = val;**
20. **}**
21. **tag[k] = 1;**
22. **}**
23. **void push\_down(int k) {**
24. **if (tag[k]) {**
25. **change(ls, laz[k]);**
26. **change(rs, laz[k]);**
27. **tag[k] = 0;**
28. **laz[k] = {};**
29. **}**
30. **}**
31. **void init(int \_n) {**
32. **n = \_n;**
33. **seg.clear(), seg.resize(n \* 4 + 100);**
34. **tag.clear(), tag.resize(n \* 4 + 100);**
35. **laz.clear(), laz.resize(n \* 4 + 100);**
36. **return;**
37. **}**
38. **void init(vector<Info> bg) {**
39. **init(bg.size() - 1);**
40. **function<void(int, int, int)> build = [&](int l, int r, int k) {**
41. **if (l == r) return (void)(seg[k] = bg[l]);**
42. **build(l, mid, ls), build(mid + 1, r, rs);**
43. **push\_up(k);**
44. **};**
45. **build(1, n, 1);**
46. **}**
47. **Info ask(int L, int R, int l, int r, int k) {**
48. **if (l > R || r < L) return Info();**
49. **if (L <= l && r <= R) {**
50. **return seg[k];**
51. **}**
52. **push\_down(k);**
53. **return ask(L, R, l, mid, ls) + ask(L, R, mid + 1, r, rs);**
54. **}**
55. **Info ask(int L, int R) {**
56. **return ask(L, R, 1, n, 1);**
57. **}**
58. **void upd(int to, Info val, int l, int r, int k) {**
59. **if (l == r) return (void)(seg[k] = val);**
60. **push\_down(k);**
61. **if (to <= mid) upd(to, val, l, mid, ls);**
62. **else upd(to, val, mid + 1, r, rs);**
63. **push\_up(k);**
64. **}**
65. **void upd(int to, Info val) {**
66. **upd(to, val, 1, n, 1);**
67. **}**
68. **void add(int L, int R, Lazy val, int l, int r, int k) {**
69. **if (l > R || r < L) return;**
70. **if (L <= l && r <= R) {**
71. **change(k, val);**
72. **return;**
73. **}**
74. **push\_down(k);**
75. **add(L, R, val, l, mid, ls);**
76. **add(L, R, val, mid + 1, r, rs);**
77. **push\_up(k);**
78. **}**
79. **void add(int L, int R, Lazy val) {**
80. **add(L, R, val, 1, n, 1);**
81. **}**
82. **int rnk; // 下面两个查询，如果查区间第k大时用，下面两个询问均返回位置**
83. **template<class F>**
84. **int findFirst(int L, int R, int l, int r, int k, F pred) {**
85. **if (l > R || r < L || !pred(seg[k])) {**
86. **if (L <= l && r <= R) rnk -= seg[k].cnt;**
87. **return -1;**
88. **}**
89. **if (l == r) {**
90. **return l;**
91. **}**
92. **push\_down(k);**
93. **int res = findFirst(L, R, l, mid, ls, pred);**
94. **if (res == -1) {**
95. **res = findFirst(L, R, mid + 1, r, rs, pred);**
96. **}**
97. **return res;**
98. **}**
99. **template<class F>**
100. **int findFirst(int l, int r, F pred, int need) {**
101. **rnk = need;**
102. **return findFirst(l, r, 1, n, 1, pred);**
103. **}**
104. **template<class F>**
105. **int findLast(int L, int R, int l, int r, int k, F pred) {**
106. **if (l > R || r < L || !pred(seg[k])) {**
107. **return -1;**
108. **}**
109. **if (l == r) {**
110. **return l;**
111. **}**
112. **push\_down(k);**
113. **int res = findLast(L, R, mid + 1, r, rs, pred);**
114. **if (res == -1) {**
115. **res = findLast(L, R, l, mid, ls, pred);**
116. **}**
117. **return res;**
118. **}**
119. **template<class F>**
120. **int findLast(int l, int r, F pred) {**
121. **return findLast(l, r, 1, n, 1, pred);**
122. **}**
123. **#undef ls**
124. **#undef rs**
125. **#undef mid**
126. **};**
127. **/\***
128. **(1) implement struct Info (2) implement struct Lazy (3) implement Info operator +(Info a, Info b) (4) implement Info operator +(Info a, Lazy b) (5) implement Lazy operator +(Lazy a, Lazy b)**
129. **.init(vector bg/size\_bg) should give vector or size**
130. **vector first element should start from 1 (index 0 not use)**
131. **\*/**

1.无旋treap

1. **mt19937 rng(time(0));**
2. **struct Node {**
3. **int fa;**
4. **int ch[2], w, rnd, laz, sz;**
5. **bool rev;**
6. **} node[N];**
8. **#define ls node[nw].ch[0]**
9. **#define rs node[nw].ch[1]**
11. **int tot;**
12. **void push\_up(int nw)**
13. **{**
14. **node[nw].sz = node[ls].sz + node[rs].sz + 1;**
15. **if(ls)node[ls].fa = nw;**
16. **if(rs)node[rs].fa = nw;**
17. **}**
18. **int new\_node(int w)**
19. **{**
20. **int nw = ++tot;**
21. **node[nw].rnd = rng();**
22. **node[nw].laz = node[nw].rev = 0;**
23. **ls = rs = 0;**
24. **node[nw].sz = 1;**
25. **node[nw].fa = 0;**
26. **node[nw].w = w;**
27. **return nw;**
28. **}**
29. **void R(int nw)**
30. **{**
31. **if (!nw) {**
32. **return;**
33. **}**
34. **swap(ls, rs);**
35. **node[nw].rev ^= 1;**
36. **}**
37. **void push\_down(int nw)**
38. **{**
39. **if (node[nw].rev) {**
40. **R(ls), R(rs);**
41. **node[nw].rev = 0;**
42. **}**
43. **}**
44. **int mg(int x, int y)**
45. **{**
46. **if (!x || !y) {**
47. **return x + y;**
48. **}**
49. **push\_down(x), push\_down(y);**
50. **if (node[x].rnd <= node[y].rnd) {**
51. **node[x].ch[1] = mg(node[x].ch[1], y);**
52. **push\_up(x);**
53. **return x;**
54. **} else {**
55. **node[y].ch[0] = mg(x, node[y].ch[0]);**
56. **push\_up(y);**
57. **return y;**
58. **}**
59. **}**
60. **void split1(int nw, int k, int &x, int &y)**
61. **{**
62. **if (!nw) {**
63. **x = y = 0;**
64. **} else {**
65. **push\_down(nw);**
66. **node[nw].fa = 0;**
67. **if (node[ls].sz >= k) {**
68. **y = nw, split1(ls, k, x, ls);**
69. **} else {**
70. **x = nw, split1(rs, k - node[ls].sz - 1, rs, y);**
71. **}**
72. **push\_up(nw);**
73. **}**
74. **}**
75. **void split2(int nw, int k, int &x, int &y)**
76. **{**
77. **if (!nw) {**
78. **x = y = 0;**
79. **} else {**
80. **push\_down(nw);**
81. **node[nw].fa = 0;**
82. **if (node[nw].w <= k) {**
83. **x = nw, split2(rs, k, rs, y);**
84. **} else {**
85. **y = nw, split2(ls, k, x, ls);**
86. **}**
87. **push\_up(nw);**
88. **}**
89. **}**
90. **void ins(int &nw, int val)**
91. **{**
92. **static int x, y;**
93. **split2(nw, val, x, y);**
94. **nw = mg(mg(x, new\_node(val)), y);**
95. **return;**
96. **}**
97. **vector<int> get\_all(int nw)**
98. **{**
99. **if (nw == 0) {**
100. **return vector<int> {};**
101. **}**
102. **push\_down(nw);**
103. **vector<int> l, r;**
104. **l = get\_all(ls);**
105. **r = get\_all(rs);**
106. **l.pb(node[nw].w);**
107. **for (auto v : r) {**
108. **l.pb(v);**
109. **}**
110. **return l;**
111. **}**
113. **int find\_rt(int nw)**
114. **{**
115. **while(node[nw].fa) {**
116. **nw = node[nw].fa;**
117. **}**
118. **return nw;**
119. **}**
121. **bool is\_rs(int nw)**
122. **{**
123. **if (!nw || nw == node[node[nw].fa].ch[0]) {**
124. **return 0;**
125. **}**
126. **return 1;**
127. **}**
129. **int get\_rnk(int nw)**
130. **{**
131. **int res = node[ls].sz + 1;**
132. **while(node[nw].fa) {**
133. **if (is\_rs(nw)) {**
134. **nw = node[nw].fa;**
135. **res += node[ls].sz + 1;**
136. **} else {**
137. **nw = node[nw].fa;**
138. **}**
139. **}**
140. **return res;**
141. **}**

2.可并堆（随机堆）

1. **int merge(int x,int y)**
2. **{**
3. **if(!x||!y)return x+y;**
4. **if(v[x]<v[y])swap(x,y);**
5. **rand()&1?ls[x]=merge(ls[x],y):rs[x]=merge(rs[x],y);**
6. **return x;**
7. **}**

3.笛卡尔树：

类似treap，建立一个有两个关键字的二叉树，满足以第一维关键字看是二叉搜索树，即左<父<右，以第二维看是堆，即孩子大于父亲，构建方法：按第一维关键字从小到大插入，通过一个保存二维关键字的单调栈维护正在构建的树的当前最右链，插入一个点时一直pop到单调栈中第一个能做当前点父亲的点，然后将之前最后pop出的当成当前点左孩子，将当前点记为栈顶的右孩子，加入单调栈中即可，复杂度O（n）

1. **void get\_tree()**
2. **{**
3. **int s=0;**
4. **for(int i=1;i<=n;i++)**
5. **{**
6. **while(s&&num[i]<=num[st[s]])**
7. **l[i]=st[s--];//pop单调栈，同时把最后pop记为左孩子**
8. **if(st[s]) r[st[s]]=i;//当前点记为栈顶右儿子**
9. **st[++s]=i;**
10. **}**
11. **}**

7.李超线段树

李超树是维护线段的线段树，线段的定义是一些y=kx + b的直线限制其横坐标取值在一个区间，李超树可以实现nlog^2地维护某个横坐标x处y最高的线段y值（如果每个线段的横坐标定义域都是整个定义域，则复杂度为nlogn）。

1. **struct Line {**
2. **ll k, b;**
3. **ll operator () (ll x)**
4. **{**
5. **return k \* x + b;**
6. **}**
7. **Line (){};**
8. **Line (ll k, ll b):k(k), b(b){};**
9. **};**
11. **const int m = 2e5 + 100;**
12. **struct lc\_tree {**
13. **#define mid ((l + r) >> 1)**
14. **#define ls (k << 1)**
15. **#define rs (k << 1 | 1)**
16. **Line seg[m + m + 100];**
17. **bool hav[m + m + 100];**
18. **VI bin;**
20. **void ins(Line x, int L, int R, int l = 1, int r = m, int k = 1)**
21. **{**
22. **if(l > R || r < L)**
23. **return;**
24. **if(L <= l && r <= R)**
25. **{**
26. **if(!hav[k])**
27. **return (void)(hav[k] = 1, seg[k] = x, bin.pb(k));**
28. **if(seg[k](mid) > x(mid))**
29. **swap(seg[k], x);**
30. **if(x(l) < seg[k](l))**
31. **ins(x, L, R, l, mid, ls);**
32. **if(x(r) < seg[k](r))**
33. **ins(x, L, R, mid + 1, r, rs);**
34. **return;**
35. **}**
36. **ins(x, L, R, l, mid, ls);**
37. **ins(x, L, R, mid + 1, rs);**
38. **return;**
39. **}**
40. **ll ask(ll x, int l = 1, int r = m, int k = 1)**
41. **{**
42. **if(!hav[k])**
43. **return 1e18;**
44. **if(l == r)**
45. **return seg[k](x);**
46. **if(x <= mid)**
47. **return min(seg[k](x), ask(x, l, mid, ls));**
48. **else**
49. **return min(seg[k](x), ask(x, mid + 1, r, rs));**
50. **}**
51. **void init()**
52. **{**
53. **memset(hav, 0, sizeof hav);**
54. **bin.clear();**
55. **}**
56. **void clear()**
57. **{**
58. **trav(v, bin)**
59. **hav[v] = 0;**
60. **bin.clear();**
61. **}**
62. **}lc;**

8.吉司机线段树

做什么区间取min/max,赋值，然后还有询问的，具体就是维护最值、最值个数、次最值（次大次小值）

1. **struct gg{**
2. **ll fi, sc;**
3. **int num, l, r;// first mx  second mx  number of first mx**
4. **ll sum = 0;**
5. **}seg[N << 2];**
7. **int n, a[N];**
9. **void push\_up(int k)**
10. **{**
11. **int ls = k << 1;**
12. **int rs = k << 1 | 1;**
13. **seg[k].num = 0;**
14. **seg[k].sum = seg[ls].sum + seg[rs].sum;**
15. **seg[k].fi = max(seg[ls].fi, seg[rs].fi);**
16. **seg[k].sc = max(seg[ls].sc, seg[rs].sc);**
17. **if(seg[ls].fi != seg[k].fi)seg[k].sc = max(seg[k].sc, seg[ls].fi);**
18. **if(seg[rs].fi != seg[k].fi)seg[k].sc = max(seg[k].sc, seg[rs].fi);**
19. **if(seg[k].fi == seg[ls].fi)seg[k].num += seg[ls].num;**
20. **if(seg[k].fi == seg[rs].fi)seg[k].num += seg[rs].num;**
21. **}**
23. **void build(int l = 1, int r = n, int k = 1)**
24. **{**
25. **seg[k].l = l;**
26. **seg[k].r = r;**
27. **if(l == r)**
28. **{**
29. **seg[k].fi = a[l];**
30. **seg[k].sc = -1;**
31. **seg[k].num = 1;**
32. **seg[k].sum = a[l];**
33. **return;**
34. **}**
35. **int mid = (l + r) >> 1;**
36. **build(l, mid, k << 1);**
37. **build(mid + 1, r, k << 1 | 1);**
38. **push\_up(k);**
39. **}**
41. **void push\_down(int k)**
42. **{**
43. **if(seg[k << 1].fi > seg[k].fi)**
44. **{**
45. **seg[k << 1].sum += 1LL \* (seg[k].fi - seg[k << 1].fi) \* seg[k << 1].num;**
46. **seg[k << 1].fi = seg[k].fi;**
47. **}**
48. **if(seg[k << 1 | 1].fi > seg[k].fi)**
49. **{**
50. **seg[k << 1 | 1].sum += 1LL \* (seg[k].fi - seg[k << 1 | 1].fi) \* seg[k << 1 | 1].num;**
51. **seg[k << 1 | 1].fi = seg[k].fi;**
52. **}**
53. **}**
55. **void upd(int L, int R, int val, int k = 1)**
56. **{**
57. **if(seg[k].fi <= val)return;**
58. **if(seg[k].fi > val && seg[k].sc <= val)**
59. **{**
60. **seg[k].sum += 1LL \* (val - seg[k].fi) \* seg[k].num;**
61. **seg[k].fi = val;**
62. **return;**
63. **}**
64. **if(seg[k].l < seg[k].r)push\_down(k);**
65. **int mid = (seg[k].l + seg[k].r) >> 1;**
66. **if(L <= mid)upd(L, R, val, k << 1);**
67. **if(R > mid)upd(L, R, val, k << 1 | 1);**
68. **push\_up(k);**
69. **}**
71. **ll ask\_mx(int L, int R, int k = 1)**
72. **{**
73. **if(seg[k].l > R || seg[k].r < L)return -1;**
74. **if(L <= seg[k].l && seg[k].r <= R)return seg[k].fi;**
75. **if(seg[k].l != seg[k].r)push\_down(k);**
76. **return max(ask\_mx(L, R, k << 1), ask\_mx(L, R, k << 1 | 1));**
77. **}**
79. **ll ask\_sum(int L, int R, int k = 1)**
80. **{**
81. **if(seg[k].l > R || seg[k].r < L)return 0;**
82. **if(L <= seg[k].l && seg[k].r <= R)return seg[k].sum;**
83. **if(seg[k].l != seg[k].r)push\_down(k);**
84. **return ask\_sum(L, R, k << 1) + ask\_sum(L, R, k << 1 | 1);**
85. **}**

# 十一、图论问题

spfa及判环、差分约束系统：

判环：存在负环则会判断是否有点入队大于等于n次即可。（n次后仍然可以松弛）。

差分约束：一堆形如a-b<=c的不等式,最后求给定未知数范围，建图跑spfa。

（1）求取最小值，那么求出最长路，那么将不等式全部化成xi – xj >= k的形式，这样建立j->i的边，权值为k的边。

(2)如果求取的是最大值，那么求取最短路，将不等式全部化成xi – xj <= k的形式, 这样建立j->i的边，权值为k的边。

(3)如果要判断差分约束系统是否存在解，一般都是判断环，选择求最短路或者最长路求解都行，只是不等式标准化时候不同，判环地话，用spfa即可，n个点中如果同一个点入队超过n次，那么即存在环。建立的图可能不联通，需要加入一个超级源点，比如求最长路时图不联通的话，加入一个点S，对其他的每个点建立一条权值为0的边图就联通了，从S点开始进行spfa判环。

1. **int spfa(int n)**
2. **{**
3. **queue<int>q;int id;**
4. **memset(dis,127,sizeof dis);**
5. **q.push(1);dis[1]=0;inq[1]=1;**
6. **while(!q.empty())**
7. **{**
8. **id=q.front();q.pop();inq[id]=0;**
9. **vis[id]++;if(vis[id]>n)return -1;**
10. **for(int i=0;i<mp[id].size();i++)**
11. **{**
12. **if(dis[mp[id][i].first]>dis[id]+mp[id][i].second)**
13. **{**
14. **dis[mp[id][i].first]=dis[id]+mp[id][i].second;**
15. **if(!inq[mp[id][i].first])q.push(mp[id][i].first),inq[mp[id][i].first]=1;**
16. **}**
17. **}**
18. **}**
19. **return dis[n];**
20. **}**

3.欧拉路：

一张无向连通图是欧拉图，当且仅当所有节点度数是偶数。

一张无向连通图有欧拉路，当且仅当只有两个点（起点终点）度数为奇数。

求欧拉回路：

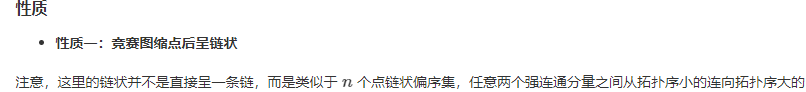
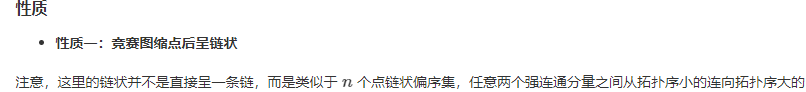
dfs（x）：

若某条边（x，y）未访问

标记为已访问，dfs（y），把y入栈。

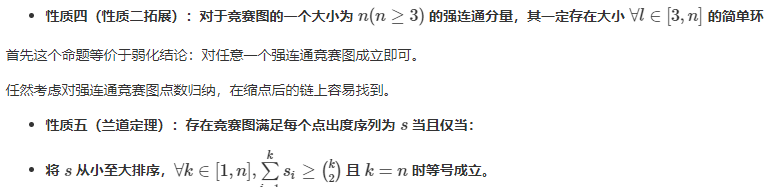
倒序输出为答案

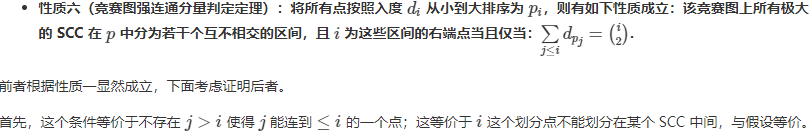
竞赛图：











4.tarjan算法总结

**（1）无向图tarjan：**

1.求low（x）：首先将low（x）初始化为dfn（x），对于连着x的边（x，y），若x为y父亲，则先dfs（y），low（x）=min（low（x），low（y）），不然说明为非树边，low（x）=min（low（x），dfn（y））

2.割点和桥

定义桥为删除掉图会分裂成两个及以上子图的边，割点为删除掉图会分裂为两个以上子图的点。若一条x到子节点y的边是桥，则要满足dfn（x）<low（y），所以dfs判一下即可，要注意与平时dfs（x）传入fa（x）不同这里要传入fa(x)到x的边标号以防重边情况。

割点情况类似，若x是割点，则要存在一个y，使得dfn（x）<=low（y），这里可以不用判断重边情况（最好还是判反正没多大区别），需要注意根是割点的条件较特殊是有两个及以上子节点。

3.点双边双：点双图是一个无向图，其中不存在割点。点双连通分量指原图一个极大点双连通子图（即不存在包含它更大点双子图）。

边双图是一个无向图，其中不存在桥。边双连通分量指原图一个极大边双连通子图（即不存在包含它更大边双子图）。

1.点双图至少满足以下两条件之一：顶点数不超过2；任意两点至少包含在一个简单环（不自交环）中。

2.一张图是边双图当且仅当任意边都包含在至少一个简单环中。

**点双连通分量BCC code:**

1. **struct BCC {**
2. **int n;**
3. **vector<vector<int>> adj;**
4. **vector<int> stk;**
5. **vector<int> dfn, low;**
6. **vector<vector<int>> bccs;**
7. **int tim, cnt;**
9. **BCC() {}**
10. **BCC(int n) {**
11. **init(n);**
12. **}**
14. **void init(int n) {**
15. **this->n = n;**
16. **adj.assign(n + 5, {});**
17. **dfn.assign(n + 5, 0);**
18. **low.resize(n + 5);**
19. **bccs.clear();**
20. **stk.clear();**
21. **tim = cnt = 0;**
22. **}**
24. **void add\_edge(int u, int v) {**
25. **adj[u].pb(v);**
26. **adj[v].pb(u);**
27. **}**
29. **void dfs(int x) {**
30. **dfn[x] = low[x] = ++tim;**
31. **stk.pb(x);**
32. **for (auto y : adj[x]) {**
33. **if (!dfn[y]) {**
34. **dfs(y);**
35. **low[x] = min(low[x], low[y]);**
36. **if (low[y] >= dfn[x]) {**
37. **vector<int> bcc;**
38. **int del;**
39. **do {**
40. **del = stk.back();**
41. **stk.pop\_back();**
42. **bcc.pb(del);**
43. **} while (del != y);**
44. **bcc.pb(x);**
45. **bccs.pb(bcc);**
46. **}**
47. **} else {**
48. **low[x] = min(low[x], dfn[y]);**
49. **}**
50. **}**
51. **}**
53. **vector<vector<int>> work() {**
54. **for (int i = 1; i <= n; i++) {**
55. **if (!dfn[i]) {**
56. **dfs(i);**
57. **if (tim == dfn[i]) { // alone**
58. **bccs.pb({i});**
59. **}**
60. **}**
61. **}**
62. **return bccs;**
63. **}**
64. **};**

边双连通分量EBCC code:

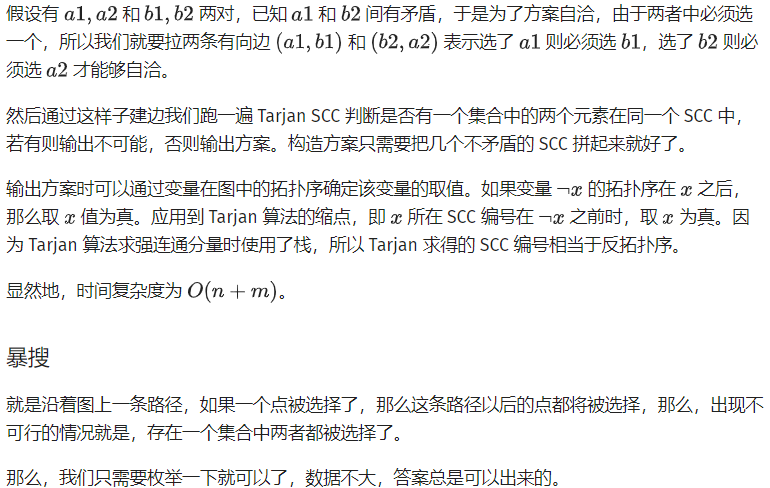
1. **struct EBCC {**
2. **int n;**
3. **vector<vector<int>> adj;**
4. **vector<int> stk;**
5. **vector<int> dfn, low, bel;**
6. **int tim, cnt;**
8. **EBCC() {}**
9. **EBCC(int n) {**
10. **init(n);**
11. **}**
13. **void init(int n) {**
14. **this->n = n;**
15. **adj.assign(n + 5, {});**
16. **dfn.assign(n + 5, 0);**
17. **low.resize(n + 5);**
18. **bel.assign(n + 5, 0);**
19. **stk.clear();**
20. **tim = cnt = 0;**
21. **}**
23. **void add\_edge(int u, int v) {**
24. **adj[u].pb(v);**
25. **adj[v].pb(u);**
26. **}**
28. **void dfs(int x, int fa) {**
29. **dfn[x] = low[x] = ++tim;**
30. **stk.pb(x);**
31. **bool skip = 0;**
32. **for (auto y : adj[x]) {**
33. **if (y == fa && !skip) {**
34. **skip = 1;**
35. **continue;**
36. **}**
37. **if (!dfn[y]) {**
38. **dfs(y, x);**
39. **low[x] = min(low[x], low[y]);**
40. **} else {**
41. **low[x] = min(low[x], dfn[y]);**
42. **}**
43. **}**
45. **if (dfn[x] == low[x]) {**
46. **int y;**
47. **++cnt;**
48. **do {**
49. **y = stk.back();**
50. **bel[y] = cnt;**
51. **stk.pop\_back();**
52. **} while (y != x);**
53. **}**
54. **}**
56. **vector<int> work() {**
57. **for (int i = 1; i <= n; i++) {**
58. **if (!dfn[i]) {**
59. **dfs(i, 0);**
60. **}**
61. **}**
62. **return bel;**
63. **}**
64. **// bel range = [1, cnt]**
65. **};**

**（2）有向图tarjan**

一张有向图，若其中任意两个点x，y都既存在x到y路径也存在y到x路径，则其为强连通图，类比点双边双分量可以定义强连通分量。求强连通分量：维护一个栈记录当前点祖先点集合，然后计算low（x）时与求割点桥差不多，就是一个点要判断是当前点祖先要看它是否在栈中。最后在x点回溯前判断是否有low（x）==dfn（x），如果有则将栈弹到x出栈为止，所有出栈的构成一个强连通分量，缩点则与边双类似。

1. **struct SCC {**
2. **int n;**
3. **vector<vector<int>> adj;**
4. **vector<int> stk;**
5. **vector<int> dfn, low, bel;**
6. **int tim, cnt;**
8. **SCC() {}**
9. **SCC(int n) {**
10. **init(n);**
11. **}**
13. **void init(int n) {**
14. **this->n = n;**
15. **adj.assign(n + 5, {});**
16. **dfn.assign(n + 5, 0);**
17. **low.resize(n + 5);**
18. **bel.assign(n + 5, 0);**
19. **stk.clear();**
20. **tim = cnt = 0;**
21. **}**
23. **void add\_edge(int u, int v) {**
24. **adj[u].pb(v);**
25. **}**
27. **void dfs(int x) {**
28. **dfn[x] = low[x] = ++tim;**
29. **stk.pb(x);**
30. **for (auto y : adj[x]) {**
31. **if (!dfn[y]) {**
32. **dfs(y);**
33. **low[x] = min(low[x], low[y]);**
34. **} else if (!bel[y]) {**
35. **low[x] = min(low[x], dfn[y]);**
36. **}**
37. **}**
39. **if (dfn[x] == low[x]) {**
40. **int y;**
41. **++cnt;**
42. **do {**
43. **y = stk.back();**
44. **bel[y] = cnt;**
45. **stk.pop\_back();**
46. **} while (y != x);**
47. **}**
48. **}**
50. **vector<int> work() {**
51. **for (int i = 1; i <= n; i++) {**
52. **if (!dfn[i]) {**
53. **dfs(i);**
54. **}**
55. **}**
56. **return bel;**
57. **}**
58. **};**

（3）2-SAT问题：



如果要输出 2-SAT 问题的一个可行解，只需要在 tarjan 缩点后所得的 DAG 上自底向上地进行选择和删除。

具体实现的时候，可以通过构造 DAG 的反图后在反图上进行拓扑排序实现；也可以根据 tarjan 缩点后，所属连通块编号越小，节点越靠近叶子节点这一性质，优先对所属连通块编号小的节点进行选择。（从小到大，之前没被分配就随便分配一个，依次往后）。

无自己到自己的2-sat可用并查集确定可行性。

**tarjan法代码**

1. **int n, m;**
2. **vector<int> adj[N];**
3. **int dfn[N], low[N], tim;**
4. **bool ins[N];**
5. **vector<int> stk;**
6. **bool hav[N];**
7. **int bel[N], num = 0;**
8. **void dfs(int x)**
9. **{**
10. **dfn[x] = low[x] = ++tim;**
11. **ins[x] = 1;**
12. **stk.pb(x);**
13. **trav(v, adj[x])**
14. **{**
15. **if(!dfn[v])**
16. **dfs(v), low[x] = min(low[x], low[v]);**
17. **else if(ins[v])**
18. **low[x] = min(low[x], dfn[v]);**
19. **}**
20. **if(low[x] == dfn[x])**
21. **{**
22. **++num;**
23. **VI bin;**
24. **while(1)**
25. **{**
26. **int nw = stk.back();**
27. **stk.pop\_back();**
28. **if(nw > n && hav[nw - n])**
29. **{**
30. **cout << "IMPOSSIBLE" << '\n';**
31. **exit(0);**
32. **}**
33. **if(nw < n && hav[nw + n])**
34. **{**
35. **cout << "IMPOSSIBLE" << '\n';**
36. **exit(0);**
37. **}**
38. **hav[nw] = 1;**
39. **ins[nw] = 0;**
40. **bel[nw] = num;**
41. **bin.pb(nw);**
42. **if(nw == x)**
43. **break;**
44. **}**
45. **trav(v, bin)**
46. **hav[v] = 0;**
47. **bin.clear();**
48. **}**
49. **}**
50. **void sol()**
51. **{**
52. **cin >> n >> m;**
53. **for(int i = 1; i <= m; i++)**
54. **{**
55. **int x, a, y, b, rx, ry;**
56. **cin >> x >> a >> y >> b;**
57. **rx = x + n, ry = y + n;**
58. **if(a == 1)**
59. **swap(x, rx);**
60. **if(b == 1)**
61. **swap(y, ry);**
62. **adj[rx].pb(y);**
63. **adj[ry].pb(x);**
64. **}**
65. **for(int i = 1; i <= n + n; i++)**
66. **if(!dfn[i])dfs(i);**
67. **cout << "POSSIBLE" << '\n';**
68. **for(int i = 1; i <= n; i++)**
69. **cout << (bel[i] < bel[i + n] ? 0 : 1) << " \n"[i == n];**
70. **}**

暴搜:

1. **struct Twosat {**
2. **int n;**
3. **vector<int> g[maxn \* 2];**
4. **bool mark[maxn \* 2];**
5. **int s[maxn \* 2], c;**
7. **bool dfs(int x) {**
8. **if (mark[x ^ 1]) return false;**
9. **if (mark[x]) return true;**
10. **mark[x] = true;**
11. **s[c++] = x;**
12. **for (int i = 0; i < (int)g[x].size(); i++)**
13. **if (!dfs(g[x][i])) return false;**
14. **return true;**
15. **}**
17. **void init(int n) {**
18. **this->n = n;**
19. **for (int i = 0; i < n \* 2; i++) g[i].clear();**
20. **memset(mark, 0, sizeof(mark));**
21. **}**
23. **void add\_clause(int x, int y) {  // 这个函数随题意变化**
24. **g[x].push\_back(y ^ 1);         // 选了 x 就必须选 y^1**
25. **g[y].push\_back(x ^ 1);**
26. **}**
28. **bool solve() {**
29. **for (int i = 0; i < n \* 2; i += 2)**
30. **if (!mark[i] && !mark[i + 1]) {**
31. **c = 0;**
32. **if (!dfs(i)) {**
33. **while (c > 0) mark[s[--c]] = false;**
34. **if (!dfs(i + 1)) return false;**
35. **}**
36. **}**
37. **return true;**
38. **}**
39. **};**

**5.生成树问题：**

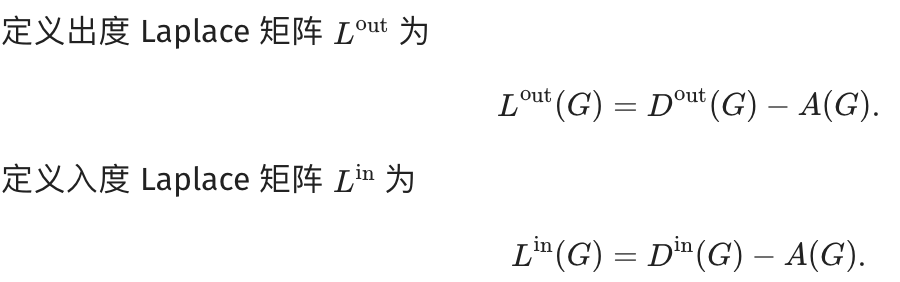
（1）克鲁斯卡尔重构树

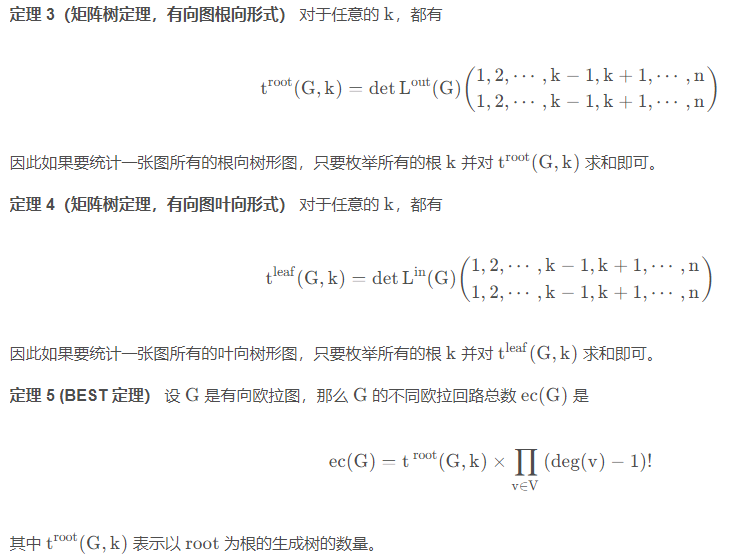
依照克鲁斯卡尔求最小生成树的做法，把边按权值从小到大排序，枚举(x,y,val)加边时，不直接加边，先访问到x,y当前并查集内的根fx,fy,然后新开一个点np,把fx和fy都挂到np下面当儿子，np点的权值则是该边边权val。这样建出来的树两点间lca的权值就是原最小生成树两点间路径边权最大值。可以解决将对边的询问转化为对点，如查询最小生成树某两个点上路径边权最大值，就可以用克鲁斯卡尔重构树重构之后，转化为欧拉序O（1）lca的问题。

（2）.生成树计数-矩阵树定理

无向图：度数矩阵减去邻接矩阵，之后任意划去一行一列，求行列式的绝对值即为所求生成树个数。

有向图：





Prufer序列：

**生成方法：**把无根树里所有度数为1的点定义为叶子，重复执行以下操作直至原树里只剩下两个节点：选取当前标号最小的叶子，把它的父亲接在当前生成的prufer序列末尾，并在原树里删除该点。**还原方法：**设一个点集V={1,2,3,...,n}，然后重复执行以下操作，取出prufer序列当前开头元素x，然后在V中从左到右遍历找到第一个没有在当前prufer序列中出现的元素y，并把x、y连一条边，然后prufer序列删除开头x，点集V删除y。直到V中剩下两个点时，把这两个点连一条边，至此，原树还原完成。**性质I**.从上面的生成还原看，prufer序列内每个点取值都可以在[1,n]内，这样可以证明一个n个点的完全图生成的无根树数量为n^(n-2) **性质II.**可以发现每个点在prufer序列每出现一次就对应一个与它相连的点的删除，所以prufer序列中某点出现次数等于该点在原树上的度数减1 **性质III.**依据上面第二条性质，假设给定树上每个节点度数，可以发现生成树个数就是一个含重复元素的集合的排列数，就是阶乘除以一堆阶乘积的东西

（3）有限制最小生成树：给定边集合黑白两色，要求求出包含恰好k条黑色边的最小生成树。用wqs二分，因为记f(k)表示选择k条黑边的答案，发现f(k)是关于k的凸函数，故二分-inf到+inf的权值，给每条黑边加上这个权值，分别求当前情况最少用多少黑边，最多用多少黑边（第二维颜色关键字两种排序方式），然后找到k所在的一段输出即可。

# 十二、dp相关

**基本思路：**用多种方式设dp状态，dp最大最小值、dp两个变量的和或差值、dp答案为某一个值的时候最多能满足多少要求……；同时正向dp不行，尝试反向（比如构造排列，然后有一些要求，可以尝试1~n插入，也可以尝试n~1插入）；dp选择的时候可以用贪心减少转移数。

0.斜率优化

dp(i) = max{a(j) \* i + b(j)}

可以直接上李超树，不过线性做法为：

维护凸包，假设，且，假如更优

得到

右侧为一个二点式确定的斜率，因此把

然后用栈、队列之类的维护上/下凸包。

wqs二分优化

一般是让你选一些东西获取最大价值，但是存在一些限制，例如只能选k个（选择方案一般不能直接求出贡献，函数较为复杂），然后会发现f(k)表示选k个获得最大价值，这东西满足凸性，即f（k + 1）- f(k) >= f(k + 2) - f(k + 1)，因此二分权值，给每选一个东西加上这个权值，此时任意选取并记录选择个数，如果选择个数的区间包含k即合法了。

决策单调性优化dp

一般dp(i)由之前某个dp(k) 转移过来，每个dp选择的最优决策点单调不下降，即用f(i)表示dp(i)选择的k，任取i < j, f(i) < f(j)。此时有两种情况：

（1）简单情况，最优决策点连续增长

（2）麻烦情况，不满足最优决策点连续增长，即g(i + 1, k) <= g(i + 1, kk)但是k与kk之间不一定是单调不降,考虑维护一个存放决策点队列，存决策点位置，以及该位置为决策点管辖的区间（二分更新）。或者**分治优化dp**：如果要求取dp(l to r)的dp值，而dp(mid)可以在O(r-l+1)复杂度内求出，那么得到dp(mid)的最优决策位置，分治解决l to mid 和 mid + 1 to r,因为此时左右区间可能决策点位置相加到O(n)，故复杂度O(nlogn).注意要保证复杂度与当前处理区间同阶，可能当前区间做完一些信息不能清空，要保留到下一层使用保证复杂度.

3.倍增优化dp

一般是维护2^k能到达的状态，然后查询的时候类似树上倍增的合并。

4.四边形不等式优化dp

**dp[i][j]=min(dp[i][k]+dp[k+1][j]+cost[i][j])**

对于一个区间值关系，如果它满足交叉小于包含的话，那么就说它是满足四边形不等式的。如当i < i' <= j < j'时，对于cost的值来说，如果它满足cost[i][j]+cost[i'][j']<=cost[i][j']+cost[i'][j] 的关系时，那么我们就说它满足四边形不等式。

优化方法：**s[i][j-1]<=s[i][j]<=s[i+1][j]，**其中s为最优决策点函数，故dp时先枚举区间长度，再枚举起点，此时最优决策点就被限制在一个区间内，可以证明复杂度O(n^2)

5.动态dp

Dp方程满足是矩阵乘积之类的，用线段树之类的维护区间信息即可（树上就是重链剖分或者全局平衡二叉树之类的）

# 十三、树上问题

1.点分治：

思想就是每次找到当前树的重心结点，然后处理这个重心下挂着各个子树间产生的对答案的贡献（也即所有包含这个重心节点的路径产生的贡献），然后递归处理每棵子树。

1. **int n;**
2. **int a[N];**
3. **vector<int> adj[N];**
4. **int sz[N], mx[N], allsz, rt;**
5. **bool vis[N];**
6. **ll res = 0, pw2[N], ipw2[N];**
8. **void get\_rt(int x, int fa) {**
9. **sz[x] = 1, mx[x] = 0;**
10. **for (auto u : adj[x]) {**
11. **if (vis[u] || u == fa) {**
12. **continue;**
13. **}**
14. **get\_rt(u, x), sz[x] += sz[u];**
15. **mx[x] = max(mx[x], sz[u]);**
16. **}**
17. **mx[x] = max(mx[x], allsz - sz[x]);**
18. **if (mx[x] < mx[rt]) {**
19. **rt = x;**
20. **}**
21. **}**
23. **int bel[N];**
24. **ll sum[N], dep[N];**
25. **vector<int> buk;**
27. **void dfs(int x, int fa) {**
28. **buk.pb(x);**
29. **for (auto u : adj[x]) {**
30. **if (vis[u] || u == fa) {**
31. **continue;**
32. **}**
33. **bel[u] = bel[x], sum[u] = 0, dep[u] = dep[x] + 1;**
34. **dfs(u, x);**
35. **}**
36. **}**
38. **void work(int x) {**
39. **vis[x] = 1, buk.clear();**
40. **buk.pb(x), bel[x] = x, sum[x] = 0, dep[x] = 0;**
41. **for (auto u : adj[x]) {**
42. **if (vis[u]) {**
43. **continue;**
44. **}**
45. **bel[u] = u, sum[u] = 0, dep[u] = 1;**
46. **dfs(u, x);**
47. **}**
48. **sort(all(buk), [&](int x, int y) {**
49. **return a[x] < a[y];**
50. **});**
51. **ll ss = 0;**
52. **for (auto nw : buk) {**
53. **int b = bel[nw];**
54. **res = (res + a[nw] \* (ss - sum[b] + mod) % mod \* ipw2[dep[nw]]) % mod;**
55. **sum[b] = (sum[b] + ipw2[dep[nw]]) % mod;**
56. **ss = (ss + ipw2[dep[nw]]) % mod;**
57. **}**
58. **ss = 0;**
59. **for (auto nw : buk) {**
60. **sum[bel[nw]] = 0;**
61. **}**
62. **reverse(all(buk));**
63. **for (auto nw : buk) {**
64. **int b = bel[nw];**
65. **res = (res - a[nw] \* (ss - sum[b] + mod) % mod \* ipw2[dep[nw]] % mod + mod) % mod;**
66. **sum[b] = (sum[b] + ipw2[dep[nw]]) % mod;**
67. **ss = (ss + ipw2[dep[nw]]) % mod;**
68. **}**
69. **ll bf = allsz;**
70. **for (auto u : adj[x]) {**
71. **if (vis[u]) {**
72. **continue;**
73. **}**
74. **allsz = (sz[u] < sz[x] ? sz[u] : bf - sz[x]);**
75. **rt = 0, get\_rt(u, x);**
76. **work(rt);**
77. **}**
78. **}**
79. **void sol()**
80. **{**
81. **cin >> n;**
82. **for (int i = 1; i < n; i++) {**
83. **int x, y;**
84. **cin >> x >> y;**
85. **adj[x].pb(y);**
86. **adj[y].pb(x);**
87. **}**
88. **for (int i = 1; i <= n; i++) {**
89. **cin >> a[i];**
90. **}**
91. **mx[0] = inf, allsz = n;**
92. **get\_rt(1, 0);**
93. **work(rt);**
94. **cout << res \* pw2[n] % mod << '\n';**
95. **}**

3.树上启发式合并：

树上启发式合并，即每次将轻儿子的信息暴力合并到重儿子的信息上从而得到自己的信息。实现的时候，不需要维护轻儿子的数组，一般我们选择全局只维护一个数组，记录上一个点的信息，然后每次暴力遍历轻儿子的子树将信息合并过来。我们只需要每次先处理了轻儿子，然后清空数组再遍历重儿子，就可以使上一个点的信息成为我们需要的重儿子的信息了。

4.虚树：

把一棵很大的树压缩一波成一棵比较小的树，然后对原树的询问就可以在这棵压缩信息的树上搞，以降低复杂度。（一般就是关键点及它们的lca）。注意一般都要把根节点先记为关键点。

1. **bool cmp(int x,int y)**
2. **{return dfn[x]<dfn[y];}**
3. **void build()   {**
4. **int tt=k,nw,f;**
5. **sort(imp+1,imp+k+1,cmp);hd=0;**
6. **for(int i=1;i<=k;i++)    {**
7. **nw=imp[i];**
8. **if(!hd){fa[nw]=0,S[++hd]=nw;continue;}**
9. **f=lca(nw,S[hd]);**
10. **while(dep[S[hd]]>dep[f])    {**
11. **if(dep[S[hd-1]]<dep[f])fa[S[hd]]=f;  //该点之前点在f上，故父亲选为f**
12. **--hd;**
13. **}**
14. **if(f!=S[hd])    {    //一个新的关键点之间的lca加入**
15. **imp[++tt]=f;**
16. **fa[f]=S[hd];**
17. **S[++hd]=f;**
18. **}**
19. **fa[nw]=f,S[++hd]=nw;**
20. **}**
21. **k=tt;sort(imp+1,imp+k+1,cmp);  //新加了关键点，重新sort**
22. **}**

# 十四、分治、分块、莫队

1.整除分块：

**Floor: x = n / l, r = n / x**

**Ceil: x = (n + l – 1) / l, r = (x == 1 ? l : (n – 1) / (x – 1))**

2.莫队：

可以理解成二维平面最小遍历的问题。变种：树上莫队，在括号序上做；回滚莫队，处理一个block的所有询问时，都把左端点重新移到块起始做。

1. **bool cmp(ques a,ques b)//排序比较函数**
2. **{return a.l/block==b.l/block?a.r<b.r:a.l/block<b.l/block;}**

其中block取根号n

**带修改莫队**：一个查询记录l,r,tim,tim表示执行这次询问修改了多少次，此时相当于三个指针移动，排序改为：

1. **bool operator < (const query &b) const {**
2. **if(bel(l) != bel(b.l)) return l < b.l;**
3. **if(bel(r) != bel(b.r)) return r < b.r;**
4. **return tim < b.tim;**
5. **}**

其中bel就是除以块大小，此时块大小取(n^(2/3)),复杂度(n^(5/3))次。

3.线段树分治：

对于一类有插入、删除（撤销插入）和整体查询操作的题目，可以考虑按时间分治（也可以叫线段树分治）。对于每一个插入操作处理出它存在的时间，就不用管删除操作了，再将这些插入操作存在区间建立一棵时间线段树，每个节点是一个vector，然后从线段树dfs到叶子经过的点上所有点vector的并就是在这个点时会对其产生影响的所有操作了。这类题不会真的要把所有vector传到叶子，可能是线性基之类的东西

4.整体二分

处理多个询问的一些问题。以区间k小为例，将询问和修改离线下来，然后对所有询问修改，二分值L、R以及Mid并同时维护哪些修改的值<=Mid,哪些询问答案会在[L,R]内，那么用树状数组跑当前这层的修改（即小于等于Mid的在bit上位置+1），然后对每个询问查询当前答案是否达到k，是的话分到[L，Mid]，不然当前k减掉k然后分到[Mid+1,R]，再把当前这层修改撤销并把修改也判断一下分到左右递归，当到达L=R时把这层的询问答案置为L即可。

# 十五、高斯消元

1. **void gauss(vector<vector<ll>> &a, int n, int m) // n个方程 m个变量，可能非满秩**
2. **{**
3. **int bg = 0;**
4. **for(int i = 0; i < m - 1; i++)**
5. **{**
6. **int ps = bg;**
7. **while(ps < n && a[ps][i] == 0)**
8. **++ps;**
9. **if(ps >= n)**
10. **continue;**
11. **if(ps != bg)**
12. **for(int j = 0; j < m; j++)**
13. **swap(a[ps][j], a[bg][j]);**
14. **ll coef = qpow(a[bg][i]);  // coef = 1 / a[bg][i]**
15. **for(int j = i; j < m; j++)**
16. **a[bg][j] = (a[bg][j] \* coef) % mod;**
17. **for(int j = 0; j < n; j++)**
18. **{**
19. **if(j == bg)**
20. **continue;**
21. **coef = a[j][i];**
22. **for(int k = i; k < m; k++)**
23. **a[j][k] = (a[j][k] - a[bg][k] \* coef % mod + mod) % mod;**
24. **}**
25. **++bg;**
26. **}**
27. **}**

# 十六、二进制技巧

**！！！！若x为long long，则使用是要改为long long版本，如\_\_builtin\_popcountll(x)**

**\_\_builtin\_ffs(x)** 返回x中最后一个为1的位是从后向前的第几位，从1开始标号，如1返回1

**\_\_builtin\_popcount(x)**：x中1的个数。

**\_\_builtin\_ctz(x)**：x末尾0的个数。x=0时结果未定义。

**\_\_builtin\_clz(x)**：x前导0的个数。x=0时结果未定义。

**\_\_builtin\_parity(x)**：x中1的奇偶性。

Lowbit(x) = x&(-x)

枚举子集的子集，3^n

1. **for (int S=1; S<(1<<n); ++S){**
2. **for (int S0=S; S0; S0=(S0-1)&S)**
3. **//do something.**
4. **}**

**bitset<1111> bst**

**bst.\_\_Find\_next(x), 找x位置之后（不包括x）下一个1的位置，失败的话返回bitset大小**

**bst.\_\_Find\_first() ，找第一个1的位置**

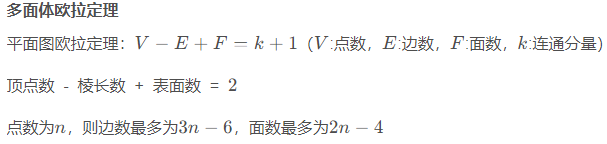
**bst.flip() 逐位取反**

**bst.count() 返回1的个数**

# 十七、计算几何常用函数

1. **const double eps = 1e-14;**
2. **const double pi = acosl(-1);**
3. **int sgn(ld x) {**
4. **return x < -eps ? -1 : x > eps;**
5. **}**
6. **int cmp(ld x, ld y) {**
7. **return sgn(x - y);**
8. **}**
10. **using T = ld;**
11. **struct Point { // 点**
12. **T x, y;**
13. **Point(T x = 0, T y = 0) : x(x), y(y) {}**
14. **bool operator<(Point B) const{ // x第一关键字**
15. **return x == B.x ? y < B.y : x < B.x;**
16. **}**
17. **bool operator==(Point B) const{**
18. **return !sgn(x - B.x) && !sgn(y - B.y);**
19. **}**
20. **bool operator <=(Point B) const {**
21. **return ((\*this) < B || (\*this) == B);**
22. **}**
23. **Point operator+(Point B) const{**
24. **return Point(x + B.x, y + B.y);**
25. **}**
26. **Point operator-(Point B) const{**
27. **return Point(x - B.x, y - B.y);**
28. **}**
29. **Point operator\*(T a) const{ // 标量乘**
30. **return Point(x \* a, y \* a);**
31. **}**
32. **Point operator/(T a) const{ // 标量除**
33. **return Point(x / a, y / a);**
34. **}**
35. **T operator\*(Point B) const{ // 点积**
36. **return x \* B.x + y \* B.y;**
37. **}**
38. **T operator^(Point B) const{ // 叉积模长**
39. **return x \* B.y - y \* B.x;**
40. **}**
41. **Point operator-() const{    // 取负,关于原点对称**
42. **return Point(-x, -y);**
43. **}**
44. **ld angle() {    // 反正切,与x轴方位角, (-pi, pi]**
45. **return atan2l(this->y, this->x);**
46. **}**
47. **T length2() { // 视为原点到(x,y)向量，模长的平方**
48. **return x \* x + y \* y;**
49. **}**
50. **ld length() { // 模长**
51. **return sqrtl(length2());**
52. **}**
53. **Point unit() {  // 单位方向向量**
54. **return \*this / this -> length();**
55. **}**
56. **Point normal() {  // 单位法向量**
57. **return Point(-y, x) / this -> length();**
58. **}**
59. **Point trunc(ld r) {  // 化为长度为r的向量**
60. **ld l = length();**
61. **if (!sgn(l))**
62. **return \*this;**
63. **r /= l;**
64. **return Point(x \* r, y \* r);**
65. **}**
66. **Point conj() { // 共轭向量**
67. **return Point(x, -y);**
68. **}**
69. **Point to\_left() { // 绕原点左转90度**
70. **return Point(-y, x);**
71. **}**
72. **Point to\_right() { // 绕原点右转90度**
73. **return Point(y, -x);**
74. **}**
75. **Point rotate(ld rad) {  // 逆时针旋转 rad 弧度**
76. **return Point(x \* cos(rad) - y \* sin(rad), x \* sin(rad) + y \* cos(rad));**
77. **}**
78. **friend int relation(Point a, Point b, Point c) { // c是否在(a,b)的逆时针侧**
79. **return sgn((b - a) ^ (c - a));**
80. **}**
81. **friend ld get\_angle(Point a, Point b) { // 向量夹角**
82. **return acosl((a \* b) / a.length() / b.length());**
83. **}**
84. **friend T area(Point a, Point b, Point c) { //**
85. **return fabsl((b - a) ^ (c - a)); // (a,b)(a,c)平行四边形面积**
86. **}**
87. **friend T get\_dis2(Point a, Point b) { // 两间距离方**
88. **return (a - b).length2();**
89. **}**
90. **friend ld get\_dis(Point a, Point b) { // 两点距离**
91. **return sqrtl(get\_dis2(a, b));**
92. **}**
93. **friend ld project(Point a, Point b, Point c) {  // 求向量ac在向量ab上的投影长度**
94. **return ((b - a) \* (c - a)) / (b - a).length();**
95. **}**
96. **bool up() const { // 是否在一二象限内,象限的定义均为左闭右开,即第一象限[0, pi/2)**
97. **return y > 0 || (y == 0 && x >= 0);**
98. **}**
99. **friend ostream& operator<<(ostream& os, Point a) {**
100. **return os << "(" << a.x << ',' << a.y << ')';**
101. **}**
102. **};**
104. **bool argcmp(Point a, Point b) { // 关于原点极角排序,一二三四象限顺序**
105. **if (a.up() != b.up())**
106. **return a.up() > b.up();**
107. **return (a ^ b) == 0 ? a.x < b.x : (a ^ b) > 0;**
108. **}**
109. **Point rotate(Point a, Point bas, ld theta) { // a点绕bas点逆时针转theta弧度**
110. **return (a - bas).rotate(theta) + bas;**
111. **}**
112. **Point reflect(Point v, Point l) { // 光线v照射到平面l后反射**
113. **Point res;**
114. **Point E = l / l.length();  // 单位向量**
115. **T d = E \* v;**
116. **return (E \* 2 \* d - v);**
117. **}**
119. **struct Line { // 直线**
120. **Point p, v; // p为直线上一点,v为方向向量**
121. **ld rad; // rad为方向角 (-pi, pi]**
122. **//亦可表示向量v逆时针方向的半平面**
123. **Line(){}**
124. **Line(Point p, Point v):p(p), v(v) {**
125. **rad = atan2l(v.y, v.x);**
126. **}**
127. **Point get\_point(ld t) {**
128. **return p + v \* t;**
129. **}**
130. **int under(Point a) { // 射线是否在点a下方**
131. **return relation(p, p + v, a);**
132. **}**
133. **bool operator <(Line b) { // 比较**
134. **if (!cmp(rad, b.rad)) {**
135. **return under(b.p) < 0; // 靠上侧的排前面**
136. **}**
137. **return rad < b.rad;**
138. **}**
139. **ld dis2point(Point a) {  // 点a到直线的距离**
140. **return fabsl((v ^ (a - p)) / v.length());**
141. **}**
142. **Point foot(Point a) { // 点在直线的投影点(垂足)**
143. **return p + v \* (v \* (a - p) / v.length2());**
144. **}**
145. **Point symmetry(Point a) { // 点关于直线对称的点(即镜面反射)**
146. **return foot(a) \* 2 - a;**
147. **}**
148. **bool parallel(Line b) { // 两直线是否平行(共线也算)**
149. **if (!sgn(v ^ b.v)) {**
150. **return 1;**
151. **}**
152. **return 0;**
153. **}**
154. **bool colinear(Line b) { // 两直线是否共线**
155. **if (parallel(b)) {**
156. **if (!sgn((b.p - p) ^ v)) {**
157. **return 1;**
158. **}**
159. **}**
160. **return 0;**
161. **}**
162. **Point intersect(Line b) {  // 两直线交点(不能平行)**
163. **assert(!parallel(b));**
164. **Point u = p - b.p;**
165. **T t = (b.v ^ u) / (v ^ b.v);**
166. **return get\_point(t);**
167. **}**
168. **};**
170. **struct Segment { // 线段**
171. **Point a, b;**
172. **Segment(){}**
173. **Segment(Point aa, Point bb) {**
174. **if (bb < aa) {**
175. **swap(aa, bb);**
176. **}**
177. **a = aa, b = bb;**
178. **}**
179. **bool isIntersect(Point p) { // 点p是否在线段上**
180. **return !sgn((a - p) ^ (b - p)) && sgn((a - p) \* (b - p)) <= 0;**
181. **}**
182. **bool parallel(Segment seg) {**
183. **return Line(a, b - a).parallel(Line(seg.a, seg.b - seg.a));**
184. **}**
185. **bool colinear(Segment seg)  { // 两线段是否共线, 有重合部分**
186. **// 只考虑seg退化的情况**
187. **if (seg.a == seg.b) {**
188. **if (Line(a, b - a).under(seg.a) == 0) {**
189. **if (a <= seg.a && seg.a <= b) {**
190. **return 1;**
191. **}**
192. **}**
193. **return 0;**
194. **}**
195. **if (Line(a, b - a).colinear(Line(seg.a, seg.b - seg.a))) {**
196. **if ((a <= seg.a && seg.a <= b) || (seg.a <= a && a <= seg.b)) {**
197. **return 1;**
198. **}**
199. **}**
200. **return 0;**
201. **}**
202. **bool isIntersect(Segment seg) { // 两线段是否相交(含端点含共线)**
203. **if (parallel(seg)) { //线段平行**
204. **return colinear(seg);**
205. **}**
206. **const Point &a1 = a, &a2 = b, &b1 = seg.a, &b2 = seg.b;**
207. **T c1 = (a2 - a1) ^ (b1 - a1), c2 = (a2 - a1) ^ (b2 - a1);**
208. **T c3 = (b2 - b1) ^ (a1 - b1), c4 = (b2 - b1) ^ (a2 - b1);**
209. **return sgn(c1) \* sgn(c2) <= 0 && sgn(c3) \* sgn(c4) <= 0;**
210. **}**
211. **bool isIntersect(Line l) { // 线段和直线是否相交 (含端点)**
212. **return l.under(a) \* l.under(b) <= 0;**
213. **}**
214. **ld dis2point(Point p) { // 点到线段距离**
215. **if (sgn((p - a) \* (b - a)) < 0 || sgn((p - b) \* (a - b)) < 0) {**
216. **return min(get\_dis(b, p), get\_dis(a, p));**
217. **}**
218. **return Line(a, b - a).dis2point(p);**
219. **}**
220. **ld dis2seg(Segment seg) {**
221. **if (isIntersect(seg)) {**
222. **return 0;**
223. **}**
224. **return min({dis2point(seg.a), dis2point(seg.b), seg.dis2point(a), seg.dis2point(b)});**
225. **}**
226. **};**
228. **ld get\_area(vector<Point> pts) {**
229. **ld res = 0;**
230. **for (int i = 1; i + 1 < pts.size(); i++) {**
231. **res += (pts[i] - pts[0]) ^ (pts[i + 1] - pts[0]);**
232. **}**
233. **return res / 2;**
234. **}**
236. **vector<Point> half\_plane\_intersect(vector<Line> lines) {**
237. **sort(all(lines));**
238. **deque<Point> pts;**
239. **deque<Line> ls;**
240. **for (auto l : lines) {**
241. **if (ls.empty()) {**
242. **ls.pb(l);**
243. **continue;**
244. **}**
245. **while(!pts.empty() && l.under(pts.back()) <= 0) {**
246. **pts.pop\_back();**
247. **ls.pop\_back();**
248. **}**
249. **while(!pts.empty() && l.under(pts[0]) <= 0) {**
250. **pts.pop\_front();**
251. **ls.pop\_front();**
252. **}**
253. **if (!sgn(ls.back().v ^ l.v)) {**
254. **if (sgn(ls.back().v \* l.v) > 0) {**
255. **continue;**
256. **} else {**
257. **return vector<Point>{};**
258. **}**
259. **}**
260. **pts.pb(l.intersect(ls.back()));**
261. **ls.pb(l);**
262. **}**
263. **while(!pts.empty() && ls[0].under(pts.back()) <= 0) {**
264. **pts.pop\_back();**
265. **ls.pop\_back();**
266. **}**
267. **if (ls.size() > 2) {**
268. **pts.push\_back(ls[0].intersect(ls.back()));**
269. **}**
270. **return vector<Point>(pts.begin(), pts.end());**
271. **}**
273. **vector<Line> get\_half\_plane(Point u, Segment v) {**
274. **if (((v.a - u) ^ (v.b - u)) == 0) {**
275. **return vector<Line>{Line(u, v.a - u), Line(v.a, u - v.a)};**
276. **}**
277. **if (((v.a - u) ^ (v.b - u)) < 0) {**
278. **swap(v.a, v.b);**
279. **}**
280. **return vector<Line> {**
281. **Line(v.a, u - v.a),**
282. **Line(v.a, v.b - v.a),**
283. **Line(u, v.b - u)**
284. **};**
285. **}**
287. **// (回转数法) 判点在多边形内外；点在多边形内返回1, 点在多边形外返回0, 点在多边形上返回-1**
288. **int point\_in\_polygon(const Point& p, vector<Point>& poly) {**
289. **int wn = 0;**
290. **int n = poly.size();**
291. **for (int i = 0; i < n; i++) {**
292. **if (Segment(poly[i], poly[(i + 1) % n]).isIntersect(p))**
293. **return -1;**
294. **int k = sgn((poly[(i + 1) % n] - poly[i]) ^ (p - poly[i]));**
295. **int d1 = sgn(poly[i].y - p.y);**
296. **int d2 = sgn(poly[(i + 1) % n].y - p.y);**
297. **if (k > 0 && d1 <= 0 && d2 > 0)**
298. **wn++;**
299. **if (k < 0 && d2 <= 0 && d1 > 0)**
300. **wn--;**
301. **}**
302. **if (wn != 0)**
303. **return 1;**
304. **return 0;**
305. **}**
307. **int inConvex(const Point& p, const vector<Point>& a) {   // a为凸包(按顺序排列), 1 内 0 外 -1边上**
308. **if (a.empty())**
309. **return false;**
310. **int l = 1, r = a.size() - 1;**
311. **while (l <= r) {**
312. **int mid = l + r >> 1;**
313. **double ls = (a[mid] - a[0]) ^ (p - a[0]);**
314. **double rs = (a[mid + 1] - a[0]) ^ (p - a[0]);**
315. **if (ls >= 0 && rs <= 0) {**
316. **int type = sgn((a[mid + 1] - a[mid]) ^ (p - a[mid]));**
317. **if (type == 0)**
318. **return -1;**
319. **else if (type == 1)**
320. **return 1;**
321. **return 0;**
322. **} else if (ls < 0) {**
323. **r = mid - 1;**
324. **} else {**
325. **l = mid + 1;**
326. **}**
327. **}**
328. **return false;**
329. **}**
331. **struct Circle {**
332. **Point p;**
333. **double r;**
334. **Circle(Point \_p = Point(0, 0), double \_r = 0) : p(\_p), r(\_r) {}**
335. **// 三角形外接圆**
336. **Circle(Point a, Point b, Point c) {**
337. **Line u = Line({(a + b) / 2}, {(b - a).rotate(pi / 2)});**
338. **Line v = Line({(a + c) / 2}, {(c - a).rotate(pi / 2)});**
339. **p = u.intersect(v);**
340. **r = get\_dis(p, a);**
341. **}**
342. **// 三角形内切圆(bool t 只是为了与外接圆区别)**
343. **Circle(Point a, Point b, Point c, bool t) {**
344. **Line u, v;**
345. **double m = atan2l(b.y - a.y, b.x - a.x), n = atan2l(c.y - a.y, c.x - a.x);**
346. **u.p = a;**
347. **u.v = u.p + Point(cos((n + m) / 2), sin((n + m) / 2));**
348. **v.p = b;**
349. **m = atan2l(a.y - b.y, a.x - b.x), n = atan2l(c.y - b.y, c.x - b.y);**
350. **v.v = v.p + Point(cos((n + m) / 2), sin((n + m) / 2));**
351. **p = u.intersect(v);**
352. **r = Line(a, b).dis2point(p);**
353. **}**
354. **bool operator==(Circle v) {**
355. **return (p == v.p) && sgn(r - v.r) == 0;**
356. **}**
357. **bool operator<(Circle v) const {**
358. **return ((p < v.p) || ((p == v.p) && sgn(r - v.r) < 0));**
359. **}**
360. **double area() {**
361. **return pi \* r \* r;**
362. **}**
363. **double length() {**
364. **return 2 \* pi \* r;**
365. **}**
366. **// 点和圆的关系        -1圆内   0圆上   1圆外**
367. **int relation(Point a) {**
368. **double dist = get\_dis(p, a);**
369. **if (sgn(dist - r) < 0)**
370. **return -1;**
371. **else if (sgn(dist - r) == 0)**
372. **return 0;**
373. **return 1;**
374. **}**
375. **// 直线和圆的关系     -1相交   0相切    1相离**
376. **int line\_relation(Line v) {**
377. **double dist = v.dis2point(p);**
378. **if (sgn(dist - r) < 0)**
379. **return -1;**
380. **else if (sgn(dist - r) == 0)**
381. **return 0;**
382. **else**
383. **return 1;**
384. **}**
385. **// 两圆的关系  5 相离   4 外切   3 相交   2内切    1 内含**
386. **int circle\_relation(Circle v) {**
387. **double dist = get\_dis(p, v.p);**
388. **if (sgn(dist - r - v.r) > 0)**
389. **return 5;**
390. **if (sgn(dist - r - v.r) == 0)**
391. **return 4;**
392. **double l = fabs(r - v.r);**
393. **if (sgn(dist - r - v.r) < 0 && sgn(dist - l) > 0)**
394. **return 3;**
395. **if (sgn(dist - l) == 0)**
396. **return 2;**
397. **if (sgn(dist - l) < 0)**
398. **return 1;**
399. **return -1;**
400. **}**
401. **// 求两个圆的交点，并返回交点个数**
402. **int cross\_circle(Circle v, Point& p1, Point& p2) {**
403. **int rel = circle\_relation(v);**
404. **if (rel == 1 || rel == 5)**
405. **return 0;**
406. **double d = get\_dis(p, v.p);**
407. **double l = (d \* d + r \* r - v.r \* v.r) / (d \* 2);**
408. **double h = sqrtl(r \* r - l \* l);**
409. **Point tmp = p + (v.p - p).trunc(l);**
410. **p1 = tmp + ((v.p - p).to\_left().trunc(h));**
411. **p2 = tmp + ((v.p - p).to\_right().trunc(h));**
412. **if (rel == 2 || rel == 4)**
413. **return 1;**
414. **return 2;**
415. **}**
416. **// 求直线和圆的交点，返回交点个数**
417. **int cross\_line(Line v, Point& p1, Point& p2) {**
418. **if ((\*this).line\_relation(v) == 1)**
419. **return 0;**
420. **Point a = v.foot(p);**
421. **double d = v.dis2point(p);**
422. **d = sqrtl(r \* r - d \* d);**
423. **if (sgn(d) == 0) {**
424. **p1 = a, p2 = a;**
425. **return 1;**
426. **}**
427. **p1 = a + v.v.trunc(d);**
428. **p2 = a - v.v.trunc(d);**
429. **return 2;**
430. **}**
431. **// 过一点作圆的切线(先判断点和圆的关系)**
432. **int tangent\_line(Point q, Line& u, Line& v) {**
433. **int x = relation(q);**
434. **if (x == -1)**
435. **return 0;**
436. **if (x == 0) {**
437. **u = Line(q, (p - q).to\_left());**
438. **v = u;**
439. **return 1;**
440. **}**
441. **double d = get\_dis(p, q);**
442. **double rad = asin(r / d);**
443. **u = Line(q, (p - q).rotate(rad));**
444. **v = Line(q, (p - q).rotate(-rad));**
445. **return 2;**
446. **}**
447. **// 求两圆相交面积**
448. **double circle\_cross\_area(Circle v) {**
449. **int rel = circle\_relation(v);**
450. **if (rel >= 4)**
451. **return 0;**
452. **if (rel <= 2)**
453. **return min(area(), v.area());**
454. **double d = get\_dis(p, v.p);**
455. **double hf = (r + v.r + d) / 2;**
456. **double ss = 2 \* sqrtl(hf \* (hf - r) \* (hf - v.r) \* (hf - d));**
457. **double a1 = acos((r \* r + d \* d - v.r \* v.r) / (2.0 \* r \* d));**
458. **a1 = a1 \* r \* r;**
459. **double a2 = acos((v.r \* v.r + d \* d - r \* r) / (2.0 \* v.r \* d));**
460. **a2 = a2 \* v.r \* v.r;**
461. **return a1 + a2 - ss;**
462. **}**
463. **// 得到过a,b两点, 半径为r1的两个圆**
464. **friend int get\_circle(Point a, Point b, double r1, Circle& c1, Circle& c2) {**
465. **Circle x(a, r1), y(b, r1);**
466. **int t = x.cross\_circle(y, c1.p, c2.p);**
467. **if (!t)**
468. **return 0;**
469. **c1.r = c2.r = r1;**
470. **return t;**
471. **}**
472. **};**
474. **template <class T>**
475. **struct convex {**
476. **vector<Point> q;**
477. **convex() {}**
478. **convex(vector<Point>& B) : q(B) {}**
479. **convex(const convex& B) : q(B.q) {}**
480. **convex& operator=(const convex& B) {**
481. **q = B.q;**
482. **return \*this;**
483. **}**
484. **Point& operator[](int x) noexcept {**
485. **return q[x];**
486. **}**
487. **int size() const {**
488. **return q.size();**
489. **}**
490. **int nxt(int x) const {**
491. **return x == size() - 1 ? 0 : x + 1;**
492. **}**
493. **int pre(int x) const {**
494. **return x == 0 ? size() - 1 : x - 1;**
495. **}**
496. **void init(vector<Point>& v) {**
497. **sort(v.begin(), v.end());**
498. **int n = v.size(), top = 0;**
499. **vector<int> st(n + 10);**
500. **for (int i = 0; i < n; i++) {**
501. **while (top > 1 && sgn((v[st[top]] - v[st[top - 1]]) ^ (v[i] - v[st[top - 1]])) <= 0)**
502. **top--;**
503. **st[++top] = i;**
504. **}**
505. **int k = top;**
506. **for (int i = n - 2; i >= 0; i--) {**
507. **while (top > k && sgn((v[st[top]] - v[st[top - 1]]) ^ (v[i] - v[st[top - 1]])) <= 0)**
508. **top--;**
509. **st[++top] = i;**
510. **}**
511. **for (int i = 1; i < top; i++)**
512. **q.push\_back(v[st[i]]);**
513. **return;**
514. **}**
515. **double get\_length() {**
516. **double res = 0;**
517. **for (int i = 0; i < size(); i++)**
518. **res += get\_dist(q[i], q[nxt(i)]);**
519. **return res;**
520. **}**
521. **T get\_area2() {**
522. **T res = 0;**
523. **for (int i = 0; i < size(); i++)**
524. **res += (q[i] ^ q[nxt(i)]);**
525. **return abs(res);**
526. **}**
527. **Point getBaryCentre() const {  // 重心**
528. **Point res(0, 0);**
529. **double are = 0;**
530. **const int sz = size();**
531. **for (int i = 1; i < sz - 1; i++) {**
532. **double tmp = (q[i] - q[0]) ^ (q[i + 1] - q[0]);**
533. **if (!sgn(tmp))**
534. **continue;**
535. **are += tmp;**
536. **res.x += (q[0].x + q[i].x + q[i + 1].x) / 3 \* tmp;**
537. **res.y += (q[0].y + q[i].y + q[i + 1].y) / 3 \* tmp;**
538. **}**
539. **if (sgn(are))**
540. **res = res / are;**
541. **return res;**
542. **}**
543. **vector<T> sum;**
544. **void get\_sum() {**
545. **vector<T> a(q.size());**
546. **for (int i = 0; i < q.size(); i++)**
547. **a[i] = q[pre(i)] ^ q[i];**
548. **sum.resize(q.size());**
549. **partial\_sum(a.begin(), a.end(), sum.begin());**
550. **}**
551. **T query\_sum(int l, int r) const {**
552. **if (l <= r)**
553. **return sum[r] - sum[l] + (q[r] ^ q[l]);**
554. **return sum[size() - 1] - sum[l] + sum[r] + (q[r] ^ q[l]);**
555. **}**
557. **// 闵可夫斯基和**
558. **convex operator+(const convex& B) const {**
559. **const auto& a = this->q;**
560. **const auto& b = B.q;**
561. **int n = q.size(), m = b.size();**
562. **Point sa = q[0], sb = b[0];**
563. **for (int i = 0; i < n; i++) {**
564. **if (a[i].y < sa.y || (a[i].y == sa.y && a[i].x < sa.x))**
565. **sa = a[i];**
566. **}**
567. **for (int i = 0; i < m; i++) {**
568. **if (b[i].y < sb.y || (b[i].y == sb.y && b[i].x < sb.x)) {**
569. **sb = b[i];**
570. **}**
571. **}**
572. **auto s = sa + sb;**
573. **vector<Point> d(n + m);**
574. **for (int i = 0; i < n; i++)**
575. **d[i] = a[(i + 1) % n] - a[i];**
576. **for (int i = 0; i < m; i++)**
577. **d[n + i] = b[(i + 1) % m] - b[i];**
578. **sort(d.begin(), d.end(), [&](const Point& A, const Point& B) {**
579. **if (A.up() ^ B.up())**
580. **return A.up() > B.up();**
581. **return (A ^ B) > 0;**
582. **});**
583. **vector<Point> c(n + m);**
584. **c[0] = s;**
585. **for (int i = 0; i < n + m - 1; i++)**
586. **c[i + 1] = c[i] + d[i];**
587. **convex res;**
588. **res.init(c);**
589. **return res;**
590. **}**
592. **// 旋转卡壳**
593. **template <class F>**
594. **void rotating\_calipres(const F& func) const {**
595. **for (int i = 0, j = 1; i < q.size(); i++) {**
596. **auto d = q[i], e = q[nxt(i)];**
597. **func(d, e, q[j]);**
598. **while (area(d, e, q[j]) <= area(d, e, q[nxt(j)])) {**
599. **j = nxt(j);**
600. **func(d, e, q[j]);**
601. **}**
602. **}**
603. **}**
605. **// 凸包直径(平方)**
606. **T diameter2() const {**
607. **if (q.size() == 1)**
608. **return 0;**
609. **if (q.size() == 2)**
610. **return get\_dis2(q[0], q[1]);**
611. **T ans = 0;**
612. **auto func = [&](const Point& a, const Point& b, const Point& c) {**
613. **ans = max({ans, get\_dis2(a, c), get\_dis2(b, c)});**
614. **};**
615. **rotating\_calipres(func);**
616. **return ans;**
617. **}**
619. **// 凸多边形关于某一方向的极点, 复杂度 O(logn)**
620. **template <class F>**
621. **int extreme(const F& dir) const {**
622. **const auto check = [&](const int i) {**
623. **return sgn(dir(q[i]) ^ (q[nxt(i)] - q[i])) >= 0;**
624. **};**
625. **const auto dir0 = dir(q[0]);**
626. **const auto check0 = check(0);**
627. **if (!check0 && check(this->size() - 1))**
628. **return 0;**
629. **const auto cmp = [&](const Point& v) {**
630. **const int vi = &v - q.data();**
631. **if (vi == 0)**
632. **return 1;**
633. **const auto checkv = check(vi);**
634. **const auto t = sgn(dir0 ^ (v - q[0]));**
635. **if (vi == 1 && checkv == check0 && sgn(dir0 ^ (v - q[0])) == 0)**
636. **return 1;**
637. **return checkv ^ (checkv == check0 && t <= 0);**
638. **};**
639. **return partition\_point(q.begin(), q.end(), cmp) - q.begin();**
640. **}**
642. **// 过凸多边形外一点求凸多边形的切线, 返回切点下标, 复杂度 O(logn)**
643. **// 必须保证点在多边形外**
644. **pair<int, int> tangent(const Point& a) const {**
645. **const int i = extreme([&](const Point& u) {**
646. **return u - a;**
647. **});**
648. **const int j = extreme([&](const Point& u) {**
649. **return a - u;**
650. **});**
651. **return {i, j};**
652. **}**
654. **// 求平行于给定直线的凸多边形的切线, 返回切点下标, 复杂度 O(logn)**
655. **pair<int, int> tangent(const Line& a) const {**
656. **const int i = extreme([&](...) {**
657. **return a.v;**
658. **});**
659. **const int j = extreme([&](...) {**
660. **return -a.v;**
661. **});**
662. **return {i, j};**
663. **}**
665. **friend int inConvex(const Point& p, const convex& c) {**
666. **return inConvex(p, c.q);**
667. **}**
668. **};**
669. **using Convex = convex<ld>;**
671. **using \_T = ld;**
672. **pair<\_T, \_T> minmax\_triangle(const vector<Point>& vec) { //最小最大三角形面积**
673. **if (vec.size() <= 2)**
674. **return {0, 0};**
675. **const \_T tmpans = abs((vec[0] - vec[1]) ^ (vec[0] - vec[2]));**
676. **\_T maxans = tmpans, minans = tmpans;**
678. **vector<pair<int, int>> evt;**
679. **evt.reserve(vec.size() \* vec.size());**
681. **for (signed i = 0; i < vec.size(); i++) {**
682. **for (signed j = 0; j < vec.size(); j++) {**
683. **if (i == j || vec[i] == vec[j])**
684. **continue;**
685. **evt.push\_back({i, j});**
686. **}**
687. **}**
688. **sort(evt.begin(), evt.end(), [&](const pair<int, int>& u, const pair<int, int>& v) {**
689. **const Point du = vec[u.second] - vec[u.first], dv = vec[v.second] - vec[v.first];**
690. **return argcmp({du.y, -du.x}, {dv.y, -dv.x});**
691. **});**
692. **vector<signed> vx(vec.size()), pos(vec.size());**
693. **for (signed i = 0; i < vec.size(); i++)**
694. **vx[i] = i;**
695. **sort(vx.begin(), vx.end(), [&](int x, int y) {**
696. **return vec[x] < vec[y];**
697. **});**
698. **for (signed i = 0; i < vx.size(); i++)**
699. **pos[vx[i]] = i;**
700. **for (auto [u, v] : evt) {**
701. **signed i = pos[u], j = pos[v];**
702. **if (i > j)**
703. **swap(u, v), swap(i, j);**
704. **const Point vecu = vec[u], vecv = vec[v];**
705. **if (i > 0)**
706. **minans = min(minans, abs((vec[vx[i - 1]] - vecu) ^ (vec[vx[i - 1]] - vecv)));**
707. **if (j < vx.size() - 1)**
708. **minans = min(minans, abs((vec[vx[j + 1]] - vecu) ^ (vec[vx[j + 1]] - vecv)));**
709. **maxans = max({maxans, abs((vec[vx[0]] - vecu) ^ (vec[vx[0]] - vecv)), abs((vec[vx.back()] - vecu) ^ (vec[vx.back()] - vecv))});**
710. **swap(vx[i], vx[j]);**
711. **pos[u] = j, pos[v] = i;**
712. **}**
713. **return {minans, maxans};**
714. **}**
715. **Circle min\_circle\_cover(vector<Point> q) {**
716. **int n = q.size();**
717. **random\_shuffle(all(q));**
718. **Circle c = (Circle){q[0], 0};**
719. **auto get\_line = [](Point a, Point b) {  // 求中垂线**
720. **return Line({(a + b) / 2}, {(b - a).rotate(-pi / 2)});**
721. **};**
722. **auto get\_circle = [&](Point a, Point b, Point c) {**
723. **Line l = get\_line(a, b), r = get\_line(a, c);**
724. **Point p = l.intersect(r);**
725. **return (Circle){p, get\_dis(p, a)};**
726. **};**
727. **for (int i = 1; i < n; i++) {    // O(n)**
728. **if (cmp(c.r, get\_dis(c.p, q[i])) < 0) {**
729. **c = {q[i], 0};**
730. **for (int j = 0; j < i; j++) {**
731. **if (cmp(c.r, get\_dis(c.p, q[j])) < 0) {**
732. **c = {(q[i] + q[j]) / 2, get\_dis(q[i], q[j]) / 2};**
733. **for (int k = 0; k < j; k++) {**
734. **if (cmp(c.r, get\_dis(c.p, q[k])) < 0)**
735. **c = get\_circle(q[i], q[j], q[k]);**
736. **}**
737. **}**
738. **}**
739. **}**
740. **}**
741. **return c;**
742. **}**

多面体欧拉定理



# 二十、代码查错

0.造小数据/大数据+对拍

1.输入输出的顺序、形式

2.变量名是否打错

3.特殊情况，0,1,2等小范围情况

4.数据范围（int -> long long -> \_\_int128）

5.检查算法思路、尝试换更简单实现方式

6.全文检查，评测错误类型可能不准确！

# 二十一、Python Template

**1. import heapq**

**2. import random**

**3. from collections import deque, defaultdict**

**4. import sys**

**5. import copy #use copy.deepcopy to copy data structure!**

**6. from functools import cmp\_to\_key**

**7. input = sys.stdin.readline**

**8. sys.setrecursionlimit(int(1e6))**

**9.**

**10. ##n = int(input())**

**11. ##a = list(map(int, input().split()))**

**12.**

**13. ##自定义排序**

**14. ##def mycmp(x, y):**

**15. ## return -1 if x > y else 1**

**16. ##**

**17. ##list.sort(a, key = cmp\_to\_key(mycmp))**

**18. ##for i in range(0, len(a)):**

**19. ## if i > 0:**

**20. ## print("",end = ' ')**

**21. ## print(a[i], end = '')**

**22. ##print("")**

**23.**

**24. ##alt3注释,alt4取消**

**25. ##F1看帮助文档**