目录

[一、编译器配置 && define的东西 + 快读板子 3](#_Toc148561843)

[二、分数规划问题 6](#_Toc148561844)

[三、多项式相关 7](#_Toc148561845)

[1.任意模数fft，加优化 7](#_Toc148561846)

[多项式模板 9](#_Toc148561847)

[另类分治fft： 11](#_Toc148561848)

[四、集合卷积fwt && SOS dp 12](#_Toc148561849)

[集合卷积： 12](#_Toc148561850)

[SOS dp 13](#_Toc148561851)

[五、字符串相关 14](#_Toc148561852)

[1.二分hash： 14](#_Toc148561853)

[2. kmp： 15](#_Toc148561854)

[3.Z函数（exkmp）： 16](#_Toc148561855)

[4.ac自动机 17](#_Toc148561856)

[5.后缀自动机 19](#_Toc148561857)

[6.后缀数组 23](#_Toc148561858)

[线性后缀数组SA-IS 24](#_Toc148561859)

[回文串，manacher算法： 27](#_Toc148561860)

[回文自动机 28](#_Toc148561861)

[最小表示法 29](#_Toc148561862)

[lyndon分解： 30](#_Toc148561863)

[六、积性函数与筛法 31](#_Toc148561864)

[常用积性函数及其迪利克雷卷积 31](#_Toc148561865)

[最短线性递推式：（bm） 36](#_Toc148561866)

[七、网络流 && 匹配问题 37](#_Toc148561867)

[最大流Dinic： 37](#_Toc148561868)

[最小费用最大流mcmf： 39](#_Toc148561869)

[网络流建图技巧： 40](#_Toc148561870)

[带花树 一般图最大匹配问题 42](#_Toc148561871)

[KM算法，二分图最大权匹配 44](#_Toc148561872)

[八、计数问题&反演容斥&组合数学 46](#_Toc148561873)

[1.卡特兰数： 46](#_Toc148561874)

[2.斯特林数： 46](#_Toc148561875)

[（4）常用斯特林数公式 48](#_Toc148561876)

[3.容斥： 48](#_Toc148561877)

[4.递推逆元： 49](#_Toc148561878)

[5.组合数前缀和 50](#_Toc148561879)

[6.组合数恒等式 50](#_Toc148561880)

[7.同构问题，burnside引理 50](#_Toc148561881)

[九、数论 50](#_Toc148561882)

[1.质因数分解： 50](#_Toc148561883)

[2.扩展欧几里得 52](#_Toc148561884)

[2.5类欧几里得算法 52](#_Toc148561885)

[3.迪利克雷前后缀和： 53](#_Toc148561886)

[4.卢卡斯定理 54](#_Toc148561887)

[5.皮克定理： 54](#_Toc148561888)

[6.欧拉定理&扩展欧拉定理 54](#_Toc148561889)

[7.扩展中国剩余定理excrt 55](#_Toc148561890)

[8.扩展拔山盖世EXBSGS 55](#_Toc148561891)

[十、数据结构 57](#_Toc148561892)

[-1.0.pbds （内置hash表&可查排名的红黑树） 57](#_Toc148561893)

[0.线段树 58](#_Toc148561894)

[1/2.Hash表 62](#_Toc148561895)

[1.无旋treap 62](#_Toc148561896)

[3.笛卡尔树： 66](#_Toc148561897)

[4.LCT 66](#_Toc148561898)

[5.K-D tree 70](#_Toc148561899)

[6.带修改主席树 73](#_Toc148561900)

[7.李超线段树 75](#_Toc148561901)

[8.吉司机线段树 76](#_Toc148561902)

[9.zkw线段树 78](#_Toc148561903)

[十一、图论问题 80](#_Toc148561904)

[迪杰斯特拉和最短路树： 80](#_Toc148561905)

[spfa及判环、差分约束系统： 81](#_Toc148561906)

[3.欧拉路： 82](#_Toc148561907)

[竞赛图： 82](#_Toc148561908)

[4.tarjan算法总结 83](#_Toc148561909)

[（3）2-SAT问题： 85](#_Toc148561910)

[（4）支配树： 88](#_Toc148561911)

[（1）.克鲁斯卡尔重构树 90](#_Toc148561912)

[（2）.生成树计数-矩阵树定理 91](#_Toc148561913)

[Prufer序列： 93](#_Toc148561914)

[最小树形图 93](#_Toc148561915)

[十二、dp相关 95](#_Toc148561916)

[0.斜率优化 96](#_Toc148561917)

[wqs二分优化 96](#_Toc148561918)

[决策单调性优化dp 96](#_Toc148561919)

[3.倍增优化dp 97](#_Toc148561920)

[4.四边形不等式优化dp 97](#_Toc148561921)

[5.动态dp 97](#_Toc148561922)

[十三、树上问题 97](#_Toc148561923)

[1.点分治： 97](#_Toc148561924)

[边分治 100](#_Toc148561925)

[3.重链剖分 100](#_Toc148561926)

[树上启发式合并： 101](#_Toc148561927)

[4.虚树： 101](#_Toc148561928)

[5树上线段树合并：（也可见Sam处代码） 102](#_Toc148561929)

[十四、分治、分块、莫队 104](#_Toc148561930)

[1. 维护序列分块： 104](#_Toc148561931)

[2.对询问分块： 104](#_Toc148561932)

[3.整除分块 104](#_Toc148561933)

[4.莫队： 105](#_Toc148561934)

[5.线段树分治： 105](#_Toc148561935)

[6.整体二分 105](#_Toc148561936)

[十五、概率期望、高斯消元 106](#_Toc148561937)

[高斯消元模板： 106](#_Toc148561938)

[十六、二进制技巧 107](#_Toc148561939)

[十七、计算几何常用函数 107](#_Toc148561940)

[多面体欧拉定理 126](#_Toc148561941)

[三维计算几何： 127](#_Toc148561942)

[十八、高精度（加减法，无负数） 131](#_Toc148561943)

[十九、python模板 133](#_Toc148561944)

[二十、代码查错 136](#_Toc148561945)

# 编译器配置 && define的东西 + 快读板子

**Devc++:**

编译时加入以下命令：-std=c++14 -DLOCAL

在连接器命令行加入以下命令：-static-libgcc -Wl,--stack=128000000

代码警告多开一点

代码补全 全关

**Codeblocks:**

Setting->editor->general setting->brace completion关掉

Setting->editor->syntax highlighting -> default淡黄色

Setting->compiler->global compiler settings->general->-std=c++11打开

Setting->compiler->global compiler settings->warnings->-wall,-weffectc++,-wextra,-pedantic,-wfatal-errors,-wfloat-equal,-wunreachable-code,-wshadow打开

**Sublime：**（可以找根目录下c++默认编译文件改）

*编译文件 Tools-Build System*

{

"shell\_cmd": "g++ -Wall -DLOCAL -std=c++17 -Wl,--stack=64000000 \"${file}\" -o \"${file\_path}/${file\_base\_name}\" && start cmd /c \"\"${file\_path}/${file\_base\_name}\" & pause\"",

"file\_regex": "^(..[^:]\*):([0-9]+):?([0-9]+)?:? (.\*)$",

"working\_dir": "${file\_path}",

"selector": "source.c++",

}

*Preference-setting*

{

"update\_check": false,

"caret\_extra\_top": 0,

"caret\_extra\_bottom":0,

"caret\_extra\_width":1,

"auto\_complete\_delay": 1000,

"font\_size": 22,

"auto\_match\_enabled": false,

"ignored\_packages":

[

"Vintage",

],

"color\_scheme": "Monokai.sublime-color-scheme",

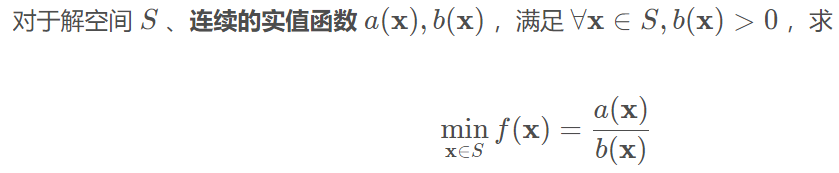
}

**define的东西 + 快读板子**

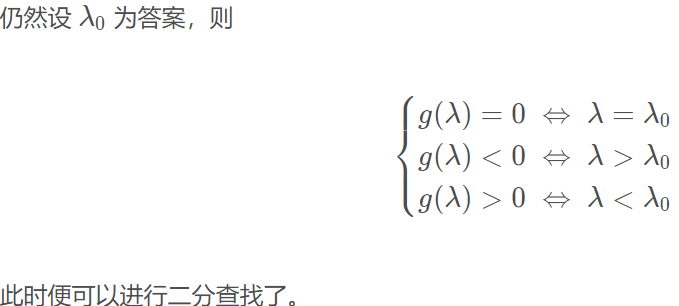
1. #include<bits/stdc++.h>
2. #define pii pair<int,int>
3. #define fi first
4. #define sc second
5. #define pb push\_back
6. #define ll long long
7. #define trav(v,x) for(auto v:x)
8. #define all(x) (x).begin(), (x).end()
9. #define VI vector<int>
10. #define VLL vector<ll>
11. #define pll pair<ll, ll>
12. #define double long double
13. template<typename T> bool chkmin(T &a, T b){return (b < a) ? a = b, 1 : 0;}
14. template<typename T> bool chkmax(T &a, T b){return (b > a) ? a = b, 1 : 0;}
15. //#define int long long
16. **using** **namespace** std;
18. **namespace** IO
19. {
20. **char** buf[1 << 20], \*p1, \*p2;
21. #define gc()                                                               \
22. (p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 20, stdin), p1 == p2) \
23. ? EOF                                                               \
24. : \*p1++)
25. **inline** **int** rd()
26. {
27. **int** x = 0, f = 1;
28. **char** c = gc();
29. **while** (!isdigit(c))
30. {
31. **if** (c == '-')
32. f = -1;
33. c = gc();
34. }
35. **while** (isdigit(c)) x = x \* 10 + (c ^ 48), c = gc();
36. **return** x \* f;
37. }
38. **char** pbuf[1 << 20], \*pp = pbuf;
39. **inline** **void** push(**const** **char** &c)
40. {
41. **if** (pp - pbuf == 1 << 20) fwrite(pbuf, 1, 1 << 20, stdout), pp = pbuf;
42. \*pp++ = c;
43. }
44. **inline** **void** write(**int** x)
45. {
46. **static** **int** sta[35];
47. **int** top = 0;
48. **do**
49. {
50. sta[top++] = x % 10, x /= 10;
51. } **while** (x);
52. **while** (top) push(sta[--top] + '0');
53. }
54. **inline** **void** outall()
55. {
56. **if**(pp - pbuf)
57. fwrite(pbuf, 1, pp - pbuf, stdout);
58. }
59. }  // namespace IO
61. **using** **namespace** IO;
62. **const** **int** N = 1e6 + 100;
63. **const** ll inf = 1e18;
64. **const** ll mod = 998244353;//1e9 + 7;
66. **void** sol()
67. {
68. **int** x;
69. x = rd();
70. write(x);
71. outall();
72. }
74. **signed** main()
75. {
76. ios::sync\_with\_stdio(0);
77. cin.tie(0);
78. //  int tt;
79. //  cin >> tt;
80. //  while(tt--)
81. sol();
82. }

# 二、分数规划问题

**分数规划问题：（平均值、中位数等问题）**



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



# 三、多项式相关

1.任意模数fft，加优化，只要做4次fft即可（两个多项式A,B要做DFT，可设多项式P，Q，令Pi = Ai + i \* Bi，Qi = Ai - i \* Bi，则可以发现

Q(ωk)= conj(P(ω−k)) = conj(P(ωL−k))

只要对P做DFT就可以同时知道Q，然后P和Q相加减得到A,B做DFT结果。

IDFT逆运算，设M(ωk)=A(ωk)+i∗B(ωk)， 对M做IDFT，然后取出实部虚部的值即是A，B的结果。）

1. **const** **int** M = 32767;
2. **const** db pi = acos(-1);
3. **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];
13. **int** len, cc, wh[N];
15. cp conj(cp a)
16. {**return** cp(a.r, -a.i);}
18. **void** fft(cp \*a, **bool** inv)
19. {
20. cp tmp;
21. **for**(**int** i = 0; i < len; i++)
22. **if**(i < wh[i])swap(a[i], a[wh[i]]);
23. **for**(**int** l = 2; l <= len; l <<= 1)
24. {
25. **int** mid = l >> 1;
26. **for**(**int** i = 0; i < len; i += l)
27. {
28. **for**(**int** j = 0; j < mid; j++)
29. {
30. tmp = a[i + j + mid] \* (inv ? w[len - len / l \* j] : w[len / l \* j]);
31. a[i + j + mid] = a[i + j] - tmp;
32. a[i + j] = a[i + j] + tmp;
33. }
34. }
35. }
36. }
38. VI mul(VI &a, VI &b) // mtt with M = 32767
39. {
40. VI res;
41. len = 1, cc = 0;
42. **while**(len < a.size() + b.size())
43. len <<= 1, ++cc;
44. **for**(**int** i = 1; i <= len; i++)
45. wh[i] = (wh[i >> 1] >> 1) | ((i & 1) << (cc - 1));
46. **for**(**int** i = 0; i <= len; i++)
47. w[i] = cp(cos(2.0 \* pi \* i / len), sin(2.0 \* pi \* i / len));
48. **int** sz = a.size() + b.size() - 1;
49. a.resize(len), b.resize(len);
50. **for**(**int** i = 0; i < len; i++)
51. {
52. A[i] = cp(a[i] / M, a[i] % M);
53. B[i] = cp(b[i] / M, b[i] % M);
54. }
55. fft(A, 0), fft(B, 0);
56. **for**(**int** i = 0; i < len; i++)
57. {
58. cp aa, bb, cc, dd;
59. **int** j = (len - i) % len;
60. aa = (A[i] + conj(A[j])) \* cp(0.5, 0);
61. bb = (A[i] - conj(A[j])) \* cp(0, -0.5);
62. cc = (B[i] + conj(B[j])) \* cp(0.5, 0);
63. dd = (B[i] - conj(B[j])) \* cp(0, -0.5);
64. AA[i] = aa \* cc + aa \* dd \* cp(0, 1);
65. BB[i] = bb \* dd + bb \* cc \* cp(0, 1);
66. }
67. fft(AA, 1), fft(BB, 1);
68. res.resize(sz);
69. **for**(**int** i = 0; i < sz; i++)
70. {
71. ll ac, ad, bc, bd;
72. ac = (ll)(AA[i].r / len + 0.5) % mod;
73. ad = (ll)(AA[i].i / len + 0.5) % mod;
74. bd = (ll)(BB[i].r / len + 0.5) % mod;
75. bc = (ll)(BB[i].i / len + 0.5) % mod;
76. res[i] = (ac \* M \* M + (ad + bc) \* M + bd) % mod;
77. }
78. **return** res;
79. }

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. }
99. **using** **namespace** Poly;

另类分治fft：

求f（x）=sigma f（i）×f（x-i）：（下面F与G都与f有关），不严谨证明：考虑左端点为1，例如[1...8]区间，那么求完[1...4]，此时考察的是两项都是左区间相乘对右边点的贡献，比如5,能发现f[1] \* f[4] 以及f[4] \* f[1]都有了，因此只要像正常分治fft做即可；左端点不是1时，比如[5..8] 中[5,6]对[7,8]贡献，可以发现此时考察的是一项为左区间另一项为之前已经算出来的值相乘对右端点贡献（因为容易证明递归到的区间[l..r]当l>1时一定有l\*2 > r），那么此时像正常分治做就只计算了一次f[7]+=f[2] \* f[5],需要额外计算一个f[7] += f[5] \* f[2]

1. **if**(l==r)
2. {
3. **if**(l==1)f[l]=1;
4. **else** Mul(f[l],jc[l-2]);
5. **return**;
6. }
7. **int** mid=(l+r)>>1;
8. sol(l,mid);
9. pre\_ntt(r-l+2);
10. **for**(**int** i=0;i<len;i++)A[i]=B[i]=0;
11. **for**(**int** i=l;i<=mid;i++)A[i-l]=F(i);//1LL\*(jc[i-1]-f[i]+mod)%mod\*ijc[i-1]%mod;
12. **for**(**int** i=1;i<=r-l;i++)B[i]=G(i);
13. ntt(A,0),ntt(B,0);
14. **for**(**int** i=0;i<len;i++)A[i]=1LL\*A[i]\*B[i]%mod;
15. ntt(A,1);
16. **for**(**int** i=mid+1;i<=r;i++)Ad(f[i],A[i-l]);
17. **if**(l>1)
18. {
19. **for**(**int** i=0;i<len;i++)A[i]=B[i]=0;
20. **for**(**int** i=l;i<=mid;i++)A[i-l]=G(i);//1LL\*(jc[i-1]-f[i]+mod)%mod\*ijc[i-1]%mod;
21. **for**(**int** i=1;i<=r-l;i++)B[i]=F(i);
22. ntt(A,0),ntt(B,0);
23. **for**(**int** i=0;i<len;i++)A[i]=1LL\*A[i]\*B[i]%mod;
24. ntt(A,1);
25. **for**(**int** i=mid+1;i<=r;i++)Ad(f[i],A[i-l]);
26. }
27. 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：

做法，类似ntt，去掉刚开始的那个swap到某一位置，然后在把变换改成如下操作：

And：正变换A[i+j]+=A[i+j+md]，逆的A[i+j]-=A[i+j+md] （与前加减后）

Or：正变换A[i+j+md]+=A[i+j],逆的A[i+j+md]-=A[i+j+md] （或后加减前）

Xor：正变换 A'[i+j+md]=A[i+j]-A[i+j+md],A'[i+j]=A[i+j]+A[i+j+md] ,逆的\*1/2(异或最像fft）

再说一下每个变换的意义，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（也可以说每个位置的值加上所有超集[所有包含它的集合]的值)，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（每个位置的值减去所有超集位置上的值）。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（每个位置的值加上子集位置的值），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（每个位置的值减去所有子集位置的值）。Xor么。。。真不知道啥意义，有大佬知道可以说下。

所以快速沃尔什变换不仅仅是正变换与逆变换组合在一起用，拆开来也是很有用的集合操作。

1. **void** And(ll \*a,**bool** inv)
2. {
3. **for**(**int** l=2,md=1;l<=len;l<<=1,md<<=1)
4. **for**(**int** i=0;i<len;i+=l)
5. **for**(**int** j=0;j<md;j++)
6. inv?Dw(a[i+j],a[i+j+md]):Ad(a[i+j],a[i+j+md]);
7. }
9. **void** Or(ll \*a,**bool** inv)
10. {
11. **for**(**int** l=2,md=1;l<=len;l<<=1,md<<=1)
12. **for**(**int** i=0;i<len;i+=l)
13. **for**(**int** j=0;j<md;j++)
14. inv?Dw(a[i+j+md],a[i+j]):Ad(a[i+j+md],a[i+j]);
15. }
17. **void** Xor(ll \*a,**bool** inv)
18. {
19. ll tp;
20. **for**(**int** l=2,md=1;l<=len;l<<=1,md<<=1)
21. **for**(**int** i=0;i<len;i+=l)
22. **for**(**int** j=0;j<md;j++)
23. {
24. tp=a[i+j+md];
25. a[i+j+md]=(a[i+j]-tp+mod)%mod;
26. Ad(a[i+j],tp);
27. **if**(inv)Mul(a[i+j],inv2),Mul(a[i+j+md],inv2);
28. }
29. }

SOS dp

*F*[*mask*]=*i*∈*mask*∑​*A*[*i*]

可以不一定是二进制，比如问题可以是给定一些十进制数，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。

（可以发现每个数贡献到它的超集一次，贡献路径是从低位到高位依次类似前缀和往后走）

代码：（以6位10进制数为例）

1. **for**(**int** i = 0, bas = 1; i < 6; i++, bas \*= 10)
2. {
3. **for**(**int** s = 0; s <= lim; s++)
4. {
5. **if**((s / bas) % 10 < 9)
6. dp[s + bas] += dp[s];
7. }
8. }

# 五、字符串相关

1.二分hash：

二分长度查询。做最长回文子串时可以正反串各hash一遍，然后枚举中间对称位，二分长度；做字典序比较的时候可以二分求出lcp长度，然后比较下一个字符。

1. **const** pll mod = pll(1e9 + 7, 1e9 + 9);
2. **const** pll bas = pll(137, 139);
3. pll operator \*(pll a, pll b) {
4. **return** pll(a.fi \* b.fi % mod.fi, a.sc \* b.sc % mod.sc);
5. }
6. pll operator +(pll a, pll b) {
7. **return** pll((a.fi + b.fi) % mod.fi, (a.sc + b.sc) % mod.sc);
8. }
9. pll operator -(pll a, pll b) {
10. **return** pll((a.fi + mod.fi - b.fi) % mod.fi, (a.sc + mod.sc - b.sc) % mod.sc);
11. }
12. **struct** Hash {
13. vector<pll> val;
14. vector<pll> pw\_bas;
15. pll mg(pll a, pll b, **int** len\_b) {
16. **return** a \* pw\_bas[len\_b] + b;
17. }
18. **int** n;
19. **void** init(string s) { // only use 1 to n - 1 (0 must be '#')
20. n = s.length() - 1;
21. val.resize(n + 2);
22. pw\_bas.resize(n + 2);
23. pw\_bas[0] = pll(1, 1);
24. **for** (**int** i = 1; i <= n; i++) {
25. pw\_bas[i] = pw\_bas[i - 1] \* bas;
26. }
27. val[0] = pll(0, 0);
28. **for** (**int** i = 1; i <= n; i++) {
29. val[i] = val[i - 1] \* bas + pll(s[i], s[i]);
30. }
31. }
32. pll ask(**int** l, **int** r) {
33. **return** val[r] - val[l - 1] \* pw\_bas[r - l + 1];
34. }
35. }hsh;

2. kmp：

用于处理对于字符串s，想知道它在另外某个串哪些位置出现的问题，先做预处理得到一个失配数组，这个数组第i位表示s的前i位后缀与前缀相同的最大长度（border），可以O（n）求。之后匹配，当匹配到s的第i位发现失配（或者完全匹配了）后，下一个有可能匹配到当前末尾的位数一定是当前这一位后缀与前缀相同的最大长度，一直跳即可。

1. **void** cal\_fail() //下标从1开始，0的话k的失配改为-1
2. {
3. **int** k = 0;
4. nxt[1] = 0;
5. **for**(**int** i=2;i<=n2;i++)
6. {
7. **while**(k && s2[k+1]!=s2[i])k=nxt[k];
8. **if**(s2[k+1]==s2[i])++k;
9. nxt[i]=k;
10. }
11. }
12. **void** pp()
13. {
14. **int** k=0;
15. **for**(**int** i=1;i<=n1;i++)
16. {
17. **while**(k && s2[k+1]!=s1[i])k=nxt[k];
18. **if**(s2[k+1]==s1[i])++k;
19. **if**(k==n2)
20. {
21. printf("%d\n",i-n2+1);
22. k=nxt[k];
23. }
24. }
25. }

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（类似manacher），匹配的时候同理。

1. **void** sol()
2. {
3. string s, t;
4. cin >> t >> s;
5. **int** n, m;
6. n = s.length();
7. m = t.length();
8. VI Z(n, 0);
9. **int** lp, rp;
10. lp = rp = 0;
11. **for**(**int** i = 1; i < n; i++)
12. {
13. **if**(i <= rp && i + Z[i - lp] - 1 < rp)
14. Z[i] = Z[i - lp];
15. **else**
16. {
17. **int** k = max(0, rp - i + 1);
18. **while**(i + k < n && s[k] == s[i + k])
19. ++k;
20. Z[i] = k;
21. }
22. **if**(i + Z[i] - 1 > rp)
23. lp = i, rp = i + Z[i] - 1;
24. }
25. Z[0] = n;
26. ll res = 0;
27. **for**(**int** i = 0; i < n; i++)
28. res ^= 1LL \* (i + 1) \* (Z[i] + 1);
29. cout << res << '\n';
30. res = 0;
31. lp = rp = -1;
32. VI ans(m, 0);
33. **for**(**int** i = 0; i < m; i++)
34. {
35. **if**(i <= rp && i + Z[i - lp] - 1 < rp)
36. ans[i] = Z[i - lp];
37. **else**
38. {
39. **int** k = max(0, rp - i + 1);
40. **while**(i + k < m && k < n && s[k] == t[i + k])
41. ++k;
42. ans[i] = k;
43. }
44. **if**(i + ans[i] - 1 > rp)
45. lp = i, rp = i + ans[i] - 1;
46. res ^= 1LL \* (i + 1) \* (ans[i] + 1);
47. }
48. cout << res << '\n';
49. }

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() {
64. }
65. }aho;
66. // usage: aho.init() aho.ins(), aho.build()

5.后缀自动机（附线段树合并模板）：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** N = 2e5 + 100;
2. **const** **int** inf = 1e9;
4. **const** **int** MAX\_SEG\_V = 5e5;
5. **template**<**class** Info>
6. **struct** SegT {
7. #define mid ((l + r) >> 1)
8. #define nls seg[nw].ls
9. #define nrs seg[nw].rs
10. **struct** Node {
11. **int** ls = 0, rs = 0;
12. Info val;
13. }seg[N \* 20];
14. **int** tot = 0;
15. **void** push\_up(**int** nw) {
16. seg[nw].val = Info{};
17. **if** (nls) seg[nw].val = seg[nw].val + seg[nls].val;
18. **if** (nrs) seg[nw].val = seg[nw].val + seg[nrs].val;
19. }
20. **int** ins(**int** to, Info val, **int** l = 1, **int** r = MAX\_SEG\_V) {
21. **int** nw = ++tot;
22. **if** (l == r) {
23. seg[nw].val = val;
24. **return** nw;
25. }
26. **if** (to <= mid) {
27. nls = ins(to, val, l, mid);
28. } **else** {
29. nrs = ins(to, val, mid + 1, r);
30. }
31. push\_up(nw);
32. **return** nw;
33. }
34. **int** mg(**int** a, **int** b) {
35. **if** (!a || !b) **return** a + b;
36. seg[a].val = seg[a].val + seg[b].val;
37. **if** (seg[a].ls || seg[b].ls) {
38. seg[a].ls = mg(seg[a].ls, seg[b].ls);
39. }
40. **if** (seg[a].rs || seg[b].rs) {
41. seg[a].rs = mg(seg[a].rs, seg[b].rs);
42. }
43. **return** a;
44. }
45. Info ask(**int** nw, **int** L, **int** R, **int** l = 1, **int** r = MAX\_SEG\_V) {
46. **if** (!nw) **return** Info{};
47. **if** (l > R || r < L) {
48. **return** Info{};
49. }
50. **if** (L <= l && r <= R) {
51. **return** seg[nw].val;
52. }
53. **return** ask(nls, L, R, l, mid) + ask(nrs, L, R, mid + 1, r);
54. }
55. #undef mid
56. #undef nls
57. #undef nrs
58. };
59. **struct** Info {
60. **int** v = 0;
61. };
62. Info operator +(Info a, Info b) {
63. **return** {a.v + b.v};
64. }
66. **const** **int** SAM\_SZ = 2e6 + 100;
67. **struct** SAM {
68. **bool** ocr[SAM\_SZ][2]; // 多串改map
69. **int** tot, las, nxt[SAM\_SZ][26], fa[SAM\_SZ], mx[SAM\_SZ], mx\_ed[SAM\_SZ];
70. vector<**int**> adj[SAM\_SZ]; // fail tree (parent tree), edge sorted in lexi order for suffix tree
71. **void** init() {
72. **for** (**int** i = 0; i <= tot; i++) {
73. memset(nxt[i], 0, **sizeof** nxt[i]);
74. fa[i] = mx[i] = 0;
75. ocr[i][0] = ocr[i][1] = 0;
76. mx\_ed[i] = -1;
77. adj[i].clear();
78. }
79. tot = 1, las = 1;
80. }
81. **int** mn(**int** p) {
82. **return** mx[fa[p]] + 1;
83. }
84. **void** ins(**int** x, **int** op = 0, **int** ps = -1) {
85. **int** p = las, np = nxt[las][x], t, nt;
86. **if** (np) { // 广义用
87. **if** (mx[p] + 1 == mx[np]) {
88. las = np;
89. ocr[np][op] = 1;
90. } **else** {
91. nt = ++tot;
92. mx[nt] = mx[p] + 1;
93. memcpy(nxt[nt], nxt[np], **sizeof** nxt[np]);
94. fa[nt] = fa[np], fa[np] = nt;
95. **while** (p && nxt[p][x] == np) {
96. nxt[p][x] = nt, p = fa[p];
97. }
98. ocr[nt][op] = 1;
99. las = nt;
100. }
101. **return**;
102. }
103. np = ++tot, mx[np] = mx[p] + 1;
104. las = np, ocr[np][op] = 1, mx\_ed[np] = ps;
105. **while** (p && !nxt[p][x]) {
106. nxt[p][x] = np;
107. p = fa[p];
108. }
109. **if** (!p) {
110. fa[np] = 1;
111. **return**;
112. }
113. t = nxt[p][x];
114. **if** (mx[t] == mx[p] + 1) {
115. fa[np] = t;
116. } **else** {
117. nt = ++tot, mx[nt] = mx[p] + 1;
118. memcpy(nxt[nt], nxt[t], **sizeof** nxt[t]);
119. fa[nt] = fa[t], fa[t] = fa[np] = nt;
120. **while** (p && nxt[p][x] == t) {
121. nxt[p][x] = nt;
122. p = fa[p];
123. }
124. }
125. }
126. **void** build\_edge(string &s) { // 出边满足后缀树字典序，需要插入反串
127. **for** (**int** i = 2; i <= tot; i++) {
128. adj[fa[i]].pb(i);
129. }
130. auto dfs = [&](auto self, **int** x) -> **void**{
131. **for** (**int** y : adj[x]) {
132. self(self, y);
133. mx\_ed[x] = max(mx\_ed[x], mx\_ed[y]);
134. }
135. };
136. dfs(dfs, 1);
137. **for** (**int** i = 1; i <= tot; i++) {
138. sort(all(adj[i]), [&](**int** x, **int** y) {
139. **int** cx = s[mx\_ed[x] - mx[i]];
140. **int** cy = s[mx\_ed[y] - mx[i]];
141. **return** cx < cy;
142. });
143. }
144. }
145. **void** build(string &s) {
146. init();
147. **for** (**int** i = 0; i < s.length(); i++) {
148. ins(s[i] - 'a', 0, i);
149. }
150. build\_edge(s);
151. }
152. vector<**int**> match(string t) {
153. vector<**int**> cur;
154. **int** nw = 1, cur\_len = 0;
155. cur.pb(nw);
156. **for** (**int** i = 0; i < t.length(); i++) {
157. **int** x = t[i] - 'a';
158. **while** (nw > 1 && !nxt[nw][x]) {
159. nw = fa[nw];
160. cur\_len = mx[nw];
161. }
162. **if** (nxt[nw][x]) nw = nxt[nw][x], cur\_len++;
163. cur.pb(nw);
164. }
165. **return** cur;
166. }
167. **void** sol(**int** n) {
168. }
169. }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;

线性后缀数组SA-IS

1. **template**<**size\_t** size>
2. **struct** SuffixArray {
3. **bool** type[size << 1];
4. **int** bucket[size], bucket1[size];
5. **int** sa[size], rk[size], ht[size];
6. **inline** **bool** isLMS(**const** **int** i, **const** **bool** \*type) { **return** i > 0 && type[i] && !type[i - 1]; }
7. **template**<**class** T>
8. **inline** **void** inducedSort(T s, **int** \*sa, **const** **int** len, **const** **int** sigma, **const** **int** bucketSize, **bool** \*type, **int** \*bucket, **int** \*cntbuf, **int** \*p) {
9. memset(bucket, 0, **sizeof**(**int**) \* sigma);
10. memset(sa, -1, **sizeof**(**int**) \* len);
11. **for** (**register** **int** i = 0; i < len; i++) bucket[s[i]]++;
12. cntbuf[0] = bucket[0];
13. **for** (**register** **int** i = 1; i < sigma; i++) cntbuf[i] = cntbuf[i - 1] + bucket[i];
14. **for** (**register** **int** i = bucketSize - 1; i >= 0; i--) sa[--cntbuf[s[p[i]]]] = p[i];
15. **for** (**register** **int** i = 1; i < sigma; i++) cntbuf[i] = cntbuf[i - 1] + bucket[i - 1];
16. **for** (**register** **int** i = 0; i < len; i++) **if** (sa[i] > 0 && !type[sa[i] - 1]) sa[cntbuf[s[sa[i] - 1]]++] = sa[i] - 1;
17. cntbuf[0] = bucket[0];
18. **for** (**register** **int** i = 1; i < sigma; i++) cntbuf[i] = cntbuf[i - 1] + bucket[i];
19. **for** (**register** **int** i = len - 1; i >= 0; i--) **if** (sa[i] > 0 && type[sa[i] - 1]) sa[--cntbuf[s[sa[i] - 1]]] = sa[i] - 1;
20. }
21. **template**<**typename** T>
22. **inline** **void** sais(T s, **int** \*sa, **int** len, **bool** \*type, **int** \*bucket, **int** \*bucket1, **int** sigma) {
23. **int** i, j, bucketSize = 0, cnt = 0, p = -1, x, \*cntbuf = bucket + sigma;
24. type[len - 1] = 1;
25. **for** (i = len - 2; i >= 0; i--) type[i] = s[i] < s[i + 1] || (s[i] == s[i + 1] && type[i + 1]);
26. **for** (i = 1; i < len; i++) **if** (type[i] && !type[i - 1]) bucket1[bucketSize++] = i;
27. inducedSort(s, sa, len, sigma, bucketSize, type, bucket, cntbuf, bucket1);
28. **for** (i = bucketSize = 0; i < len; i++) **if** (isLMS(sa[i], type)) sa[bucketSize++] = sa[i];
29. **for** (i = bucketSize; i < len; i++) sa[i] = -1;
30. **for** (i = 0; i < bucketSize; i++) {
31. x = sa[i];
32. **for** (j = 0; j < len; j++) {
33. **if** (p == -1 || s[x + j] != s[p + j] || type[x + j] != type[p + j]) { cnt++, p = x; **break**; }
34. **else** **if** (j > 0 && (isLMS(x + j, type) || isLMS(p + j, type))) **break**;
35. }
36. x = (~x & 1 ? x >> 1 : x - 1 >> 1), sa[bucketSize + x] = cnt - 1;
37. }
38. **for** (i = j = len - 1; i >= bucketSize; i--) **if** (sa[i] >= 0) sa[j--] = sa[i];
39. **int** \*s1 = sa + len - bucketSize, \*bucket2 = bucket1 + bucketSize;
40. **if** (cnt < bucketSize) sais(s1, sa, bucketSize, type + len, bucket, bucket1 + bucketSize, cnt);
41. **else** **for** (i = 0; i < bucketSize; i++) sa[s1[i]] = i;
42. **for** (i = 0; i < bucketSize; i++) bucket2[i] = bucket1[sa[i]];
43. inducedSort(s, sa, len, sigma, bucketSize, type, bucket, cntbuf, bucket2);
44. }
45. **template**<**class** T>
46. **inline** **void** getHeight(T s, **const** **int** len, **const** **int** \*sa) {
47. **for** (**int** i = 0, k = 0; i < len; i++) {
48. **if** (rk[i] == 0) k = 0;
49. **else** {
50. **if** (k > 0) k--;
51. **int** j = sa[rk[i] - 1];
52. **while** (i + k < len && j + k < len && s[i + k] == s[j + k]) k++;
53. }
54. ht[rk[i]] = k;
55. }
56. }
57. **template**<**class** T>
58. **inline** **void** init(T s, **const** **int** len, **const** **int** sigma) {
59. sais(s, sa, len, type, bucket, bucket1, sigma);
60. **for** (**register** **int** i = 1; i < len; i++) rk[sa[i]] = i;
61. getHeight(s, len, sa);
62. }
63. };
65. SuffixArray<N> sf;
67. string s;
69. **void** sol()
70. {
71. cin >> s;
72. **int** n = s.length();
73. sf.init(s, n + 1, 256);
74. **for** (**int** i = 1; i <= n; i++) { // sa数组下标从1开始，rk从0开始，值域均[0,n-1]
75. cout << sf.sa[i] + 1 << ' ';
76. }
77. cout << '\n';
78. **for** (**int** i = 0; i < n; i++) {
79. cout << sf.rk[i] << ' ';
80. }
81. cout << '\n';
82. **for** (**int** i = 2; i <= n; i++) { // lcp(sa[i], sa[i - 1])
83. cout << sf.ht[i] << ' ';
84. }
85. }

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，见多项式相关。

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。

询问排名问题，考虑后缀数组倍增法思想。

回文串，manacher算法：

1. **int** hw[32000010];
2. string s="%#",tp;
3. **void** car()
4. {
5. **int** pos=0,mx=0;
6. **for**(**int** i=1;i<=s.length()-2;i++)
7. {
8. **if**(mx>i)
9. hw[i]=min(hw[2\*pos-i],mx-i);
10. **else** hw[i]=1;
11. **for**(;s[i-hw[i]]==s[i+hw[i]];)++hw[i];
12. **if**(hw[i]>hw[pos])
13. {
14. pos=i;
15. mx=hw[i]+i;
16. }
17. }
18. cout<<hw[pos]-1;
19. }
20. **int** main()
21. {
22. cin>>tp;
23. **for**(**int** i=0;i<tp.length();i++)
24. {
25. s+=tp[i];
26. s+='#';
27. }
28. s+='$';
29. car();
30. }

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，转移边表示同时在串开头结尾加一个字符，偶根的fail往奇根跳，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. {
10. **int** nw = las;
11. **while**(ps - len[nw] - 1 < 0 || c != s[ps - len[nw] - 1] - 'a')  {
12. nw = fail[nw];
13. }
14. **if**(nxt[nw][c]) {
15. las = nxt[nw][c];
16. ed[las].pb(ps);
17. ocr[las]++;
18. **return**;
19. }
20. **int** np = ++tot;
21. len[np] = len[nw] + 2;
22. **int** t = fail[nw];
23. **while**(ps - len[t] - 1 < 0 || c != s[ps - len[t] - 1] - 'a') {
24. t = fail[t];
25. }
26. fail[np] = nxt[t][c];
27. nxt[nw][c] = np;
28. las = np;
29. ed[np].pb(ps);
31. dep[las] = dep[fail[las]] + 1;
32. ocr[las]++;
33. }
34. **void** init(string \_s, **int** \_n)
35. {
36. s = \_s, n = \_n;
37. tot = 1;
38. len[1] = -1;
39. len[0] = 0;
40. dis[0] = dis[1] = 0;
41. fail[0] = 1;
42. dep[0] = dep[1] = 0;
43. las = 0;
44. **for** (**int** i = 0; i < n; i++) {
45. ins(s[i] - 'a', i);
46. }
47. }
48. **void** sol() {
49. }
50. }pam;

最小表示法，找到字符串循环同构字典序最小串：（可二分哈希）

1).s[i+k]>s[j+k], 由于s[i~ i+k-1 ]都不会是循环字符串的"最小表示"的前缀，i滑动到i+k+1处。

2).s[i+k]<s[j+k]，同理关系 1)，j滑动到 j+k+1 处。

3).s[i+k]==s[j+k]，则k++。

若滑动后i==j，将正在变化的那个指针在+1.直到i， j把整个字串都检验完毕，返回两者中小于len的值。如果 k==len，则返回 min(i，j)

1. **int** Min\_show()
2. {
3. **int** i=0,j=1,k=0;
4. **while**(i<n&&j<n&&k<n)
5. {
6. **if**(A[(i+k)%n]==A[(j+k)%n])
7. k++;
8. **else**
9. {
10. **if**(A[(i+k)%n]>A[(j+k)%n])
11. i+=k+1;
12. **else** j+=k+1;
13. **if**(i==j)i++;
14. k=0;
15. }
16. }
17. **return** min(i,j);
18. }

lyndon分解：

lyndon分解就是说每一个字符串，都能唯一地划分为一系列子串s1,s2,s3...sk，满足如下两条性质：

1）si是lyndon串，lyndon串的定义为最小后缀为自身的串,如ab、aab是，aa，ba不是

2) 划分出的子串字典序单调不降（即非严格增大）。

如aa可划分为a|a, bcbca可划分为bc|bc|a

求这个分解，如果想偷懒，显然可以用广为人知的后缀数组，每次把还存在的最小后缀位置作为一个划分位置即可（事实上也可以以此说明lyndon分解的存在性，唯一性的话要用反证法来证）

不过也有专门的算法，且是递推地构造，在有些题目里有用。考虑维护已划分出的最后一个lyndon分解结尾i, 当前正在构造的lydon分解上一循环节位置j，当前正在构造的lyndon分解末尾位置k，然后根据j,k 字典序关系处理各种情况即可。

1. **void** sol()
2. {
3. string s;
4. cin >> s;
5. **int** n = s.length();
7. **int** i = -1, j = 0, k = 1;
8. **int** ans = 0;
9. **while**(i < n - 1)
10. {
11. **if**(k < n && s[k] > s[j])
12. {
13. j = i + 1;
14. ++k;
15. }
16. **else** **if**(k < n && s[k] == s[j])
17. {
18. ++j, ++k;
19. }
20. **else**
21. {
22. **int** stp = k - j;
23. //  cerr << i << ' ' << j << ' ' << k << '\n';
24. **while**(i < j)
25. {
26. i += stp;
27. cerr << i + 1 << '\n';
28. ans ^= (i + 1);
29. }
30. j = i + 1;
31. k = j + 1;
32. }
33. }
34. cout << ans << '\n';
35. }

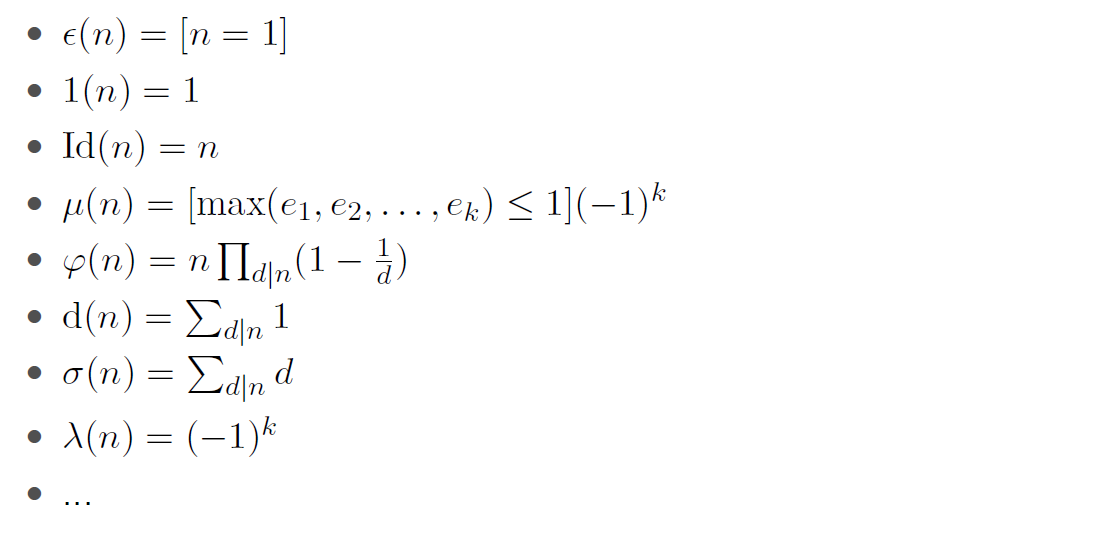
# 六、积性函数与筛法

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

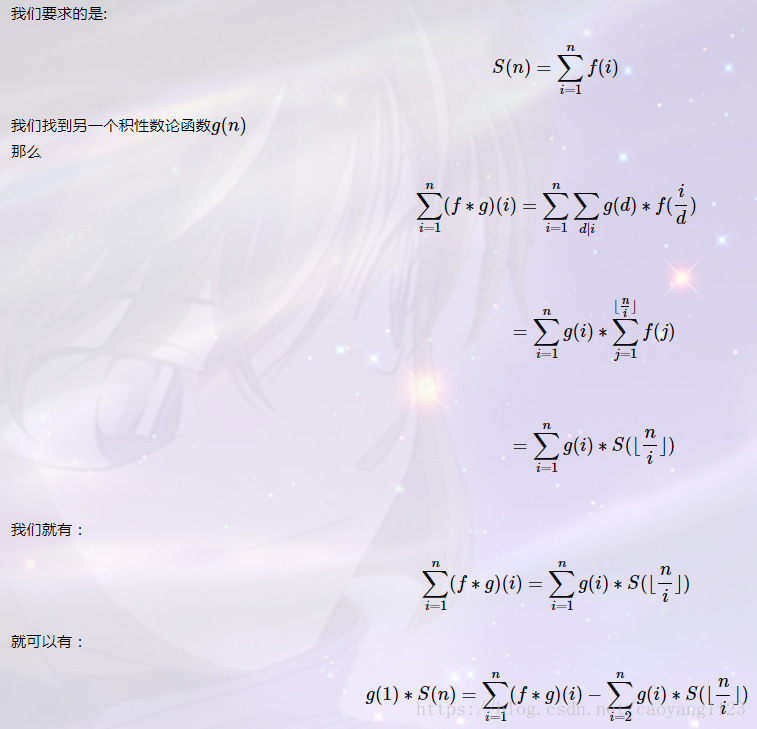
迪利克雷卷积：

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

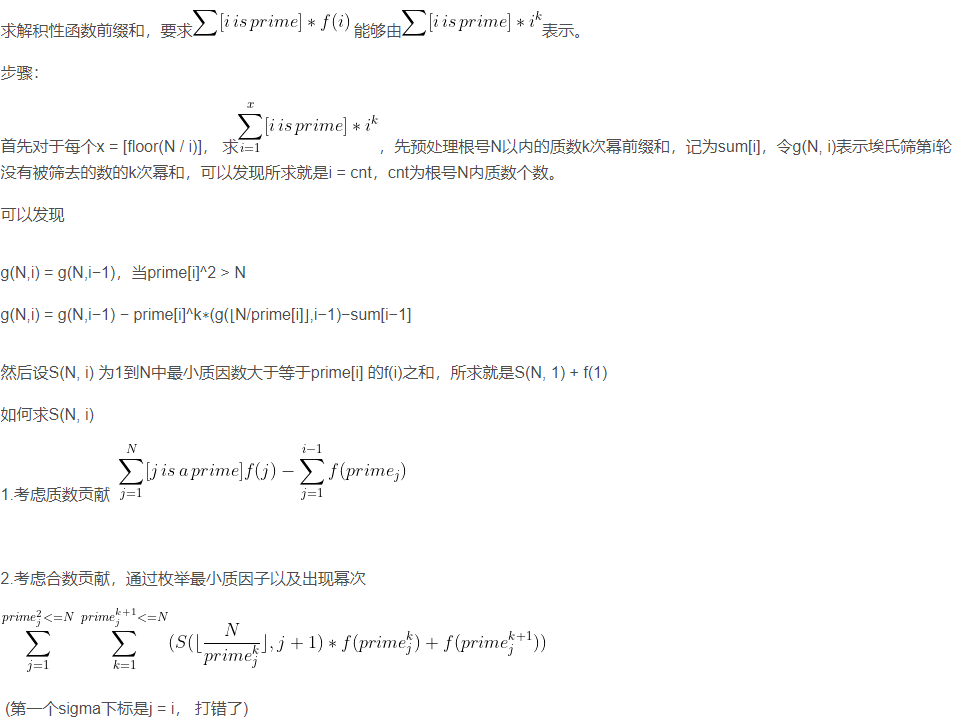


μ∗1=ϵ【莫比乌斯反演】【μ与1互为逆元】ϕ∗1=Id ，ϕ=Id∗μ ，d=1∗1， 1=μ∗d



以上为杜教筛，需要找到合适的g，满足f\*g前缀和能够快速计算。预处理出积性函数f(n)的前O(n^(2/3))项，就可以在记忆化搜索在O(n^(2/3))的复杂度计算出S(n)

1. **int** ask(**int** x)
2. {
3. **if**(x<=N)**return** sum[x];
4. **int** tp=vi[x];
5. **if**(tp)**return** tp;
6. tp=1;
7. **for**(**int** l=2,r;l<=x;l=r+1)
8. {
9. r=x/(x/l);
10. tp-=ask(x/l)\*(r-l+1);
11. }
12. **return** vi[x]=tp;
13. }
14. **void** init()
15. {
16. miu[1]=1;
17. **for**(**int** i=2;i<=N;i++)
18. {
19. **if**(!vis[i])miu[i]=-1,prime[++pnum]=i;
20. **for**(**int** j=1;j<=pnum&&prime[j]\*i<=N;j++)
21. {
22. vis[i\*prime[j]]=1;
23. **if**(i%prime[j]==0)**break**;
24. miu[i\*prime[j]]=-miu[i];
25. }
26. }
27. **for**(**int** i=1;i<=N;i++)
28. sum[i]=sum[i-1]+miu[i];
29. }



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. {
3. **const** **int** N = 2e5 + 100;
4. ll n, B, num;
5. ll val[N], g[2][N], sum[2][N], f[N], pre[N];
6. **int** ps1[N], ps2[N];
8. **int** pos(ll x)
9. {
10. **if**(x <= B)
11. **return** ps1[x];
12. **return** ps2[n / x];
13. }
15. **int** pnum;
16. **int** pri[N];
17. **bool** np[N];
19. **void** work(ll \_n)
20. {
21. n = \_n;
22. B = sqrt(n);
23. /\*
24. sum[k][i] = Prefix Sum of [pri\_i ^ k]
25. \*/
26. **for**(**int** i = 2; i <= B; i++)
27. {
28. **if**(!np[i])
29. {
30. pri[++pnum] = i;
31. sum[0][pnum] = pnum;
32. sum[1][pnum] = (sum[1][pnum - 1] + i) % mod;
33. }
34. **for**(**int** j = 1; j <= pnum && i \* pri[j] <= B; j++)
35. {
36. np[i \* pri[j]] = 1;
37. **if**(i % pri[j] == 0)
38. **break**;
39. }
40. }
41. /\*
42. Give index to Each Block of [n / d]
43. Small Index -> Bigger Original Value
45. g[k][i] Indicates Prefix Sum of (i ^ k), Notice it Starts From Index 2nd
46. \*/
47. **for**(ll l = 1, r; l <= n; l = r + 1)
48. {
49. ll x = n / l;
50. r = n / x;
51. val[++num] = x;
52. **if**(x <= B)
53. ps1[x] = num;
54. **else** ps2[n / x] = num;
55. x %= mod;
56. g[0][num] = (x - 1 + mod) % mod;
57. g[1][num] = (x - 1 + mod) % mod \* (x + 2) / 2 % mod;
58. }
59. /\*
60. Calculate Prefix Sum of (i ^ k) with [i is Prime]
61. g[k][num] = Prefix Sum from (1 to val[num]) of (i ^ k) with [i is Prime]
62. \*/
63. **for**(**int** i = 1; i <= pnum; ++i)
64. {
65. **for**(**int** j = 1; j <= num && 1LL \* pri[i] \* pri[i] <= val[j]; ++j)
66. {
67. **int** bf = pos(val[j] / pri[i]);
68. g[0][j] = (g[0][j] - g[0][bf] + sum[0][i - 1] + mod) % mod;
69. g[1][j] = (g[1][j] - pri[i] \* (g[1][bf] - sum[1][i - 1] + mod) % mod + mod) % mod;
70. }
71. }
72. /\*
73. Calculate the contribution of Prime Position to Final answer
74. f[i] = the contribution of Prime Position to Final answer in [1 to val[i]]
76. Calculate Pre[i] = Contribution of the [1st Prime to i'th Prime]
77. \*/
78. **for**(**int** i = 1; i <= num; i++)
79. f[i] = (g[1][i] - g[0][i] + mod) % mod;
80. **for**(**int** i = 1; i <= pnum; i++)
81. pre[i] = (sum[1][i] - sum[0][i] + mod) % mod;
82. **for**(**int** j = pnum; j >= 1; j--)
83. {
84. **for**(**int** i = 1; i <= num; i++)
85. {
86. **if**(1LL \* pri[j] \* pri[j] > val[i])
87. **break**;
88. ll tmp = pri[j];
89. **for**(**int** e = 1; tmp <= val[i] / pri[j]; e++, tmp \*= pri[j])
90. {
91. /\*
92. s = Contribution of [Prime\_{j} ^ e]
93. t = Contribution of [Prime\_{j} ^ (e+1)]
94. \*/
95. ll s = (tmp - tmp / pri[j]) % mod;
96. ll t = (tmp \* pri[j] - tmp) % mod;
97. (f[i] += (s \* (f[pos(val[i] / tmp)] - pre[j] + mod) + t)) %= mod;
98. }
99. }
100. }
101. // 1 should be replaced by Contribution of index 1
102. cout << (f[1] + 1) % mod;
103. }
104. }

递推f，可以求出所有n/i下取整处的积性函数前缀和值，f(i, j)表示且x为素数或者R(x) >= pri(j), R(x）表示x的最小质因子，则f(i, pnum + 1) = g(1, i)简写为g(i)

从f(i, j) 转移到 f(i + 1, j),枚举pri(j)出现次数e，式子=(f(i / pri(j) ^ e) – sum[j]) \* f(pri[j] ^ e),再补上质数幂处函数值f(pri(j) ^ (e + 1))即可

最短线性递推式：（bm）

求一个r数组，, 且m最短

Ri表示第i次修改后的最短线性递推式，且R0 = { }

当前递推式,到ai处出错，则令

修改：

1.若cnt = 0, 则

2.cnt > 0 , 令

然后即得到

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.

4.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…

5.上下界网络流：

上下界网络流本质是给流量网络的每一条边设置了流量上界 c(u,v) 和流量下界 b(u,v)。也就是说，一种可行的流必须满足 b(u,v) <= f(u,v) <= c(u,v)。同时必须满足除了源点和汇点之外的其余点流量平衡。

根据题目要求，我们可以使用上下界网络流解决不同问题。

无源汇上下界可行流

给定无源汇流量网络G。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时每一个点流量平衡。

不妨假设每条边已经流了 b(u,v) 的流量，设其为初始流。同时我们在新图中加入 u 连向 v 的流量为 c(u,v) - b(u,v) 的边。考虑在新图上进行调整。

由于最大流需要满足初始流量平衡条件（最大流可以看成是下界为 0 的上下界最大流），但是构造出来的初始流很有可能不满足初始流量平衡。假设一个点初始流入流量减初始流出流量为 M。

若 M=0，此时流量平衡，不需要附加边。

若 M>0，此时入流量过大，需要新建附加源点 S'，S' 向其连流量为 M 的附加边。

若 M<0，此时出流量过大，需要新建附加汇点 T'，其向 T' 连流量为 -M 的附加边。

如果附加边满流，说明这一个点的流量平衡条件可以满足，否则这个点的流量平衡条件不满足。（因为原图加上附加流之后才会满足原图中的流量平衡。）

在建图完毕之后跑 S' 到 T' 的最大流，若 S' 连出去的边全部满流，则存在可行流，否则不存在。

有源汇上下界可行流

给定有源汇流量网络 G。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。

假设源点为 S，汇点为 T。

则我们可以加入一条 T 到 S 的上界为 inf，下界为 0 的边转化为无源汇上下界可行流问题。

若有解，则 S 到 T 的可行流流量等于 T 到 S 的附加边的流量。 (S, T 与 S', T'不同)

有源汇上下界最大流:

给定有源汇流量网络 G。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。如果存在，询问满足标定的最大流量。

我们找到网络上的任意一个可行流。如果找不到解就可以直接结束。

否则我们考虑删去所有附加边之后的残量网络并且在网络上进行调整。（即S'，T'的那些边，以及S和T的inf边，把这些边的正边反边的las全赋为0）

我们在残量网络上再跑一次 S 到 T 的最大流，将可行流流量和最大流流量相加即为答案。

一个非常易错的问题:千万不可以在原来的流量网络上跑。

有源汇上下界最小流:

给定有源汇流量网络 G。询问是否存在一种标定每条边流量的方式，使得每条边流量满足上下界同时除了源点和汇点每一个点流量平衡。如果存在，询问满足标定的最小流量。

类似的，我们考虑将残量网络中不需要的流退掉。

我们找到网络上的任意一个可行流。如果找不到解就可以直接结束。

否则我们考虑删去所有附加边之后的残量网络。

我们在残量网络上再跑一次 T 到 S 的最大流，将可行流流量减去最大流流量即为答案。

带花树 一般图最大匹配问题

1. **int** tot, m;
2. VI adj[N];
3. **int** match[N], fa[N], vis[N], pre[N];
4. queue<**int**>que;
6. **int** find(**int** x)
7. {**return** fa[x] == x ? x : fa[x] = find(fa[x]);}
9. **int** calc\_lca(**int** x, **int** y)
10. {
11. **static** **int** buk[N];
12. **static** **int** nw;
13. ++nw;
14. **while**(1)
15. {
16. **if**(x)
17. {
18. x = find(x);
19. **if**(buk[x] == nw)
20. **return** x;
21. buk[x] = nw;
22. x = pre[match[x]];
23. }
24. swap(x, y);
25. }
26. }
28. **void** shrink(**int** x, **int** y, **int** lca)
29. {
30. **while**(find(x) != lca)
31. {
32. pre[x] = y, y = match[x];
33. **if**(vis[y] == 2)
34. {
35. vis[y] = 1;
36. que.push(y);
37. }
38. **if**(find(x) == x)
39. fa[x] = lca;
40. **if**(find(y) == y)
41. fa[y] = lca;
42. x = pre[y];
43. }
44. }
46. **bool** aug(**int** s)
47. {
48. **for**(**int** i = 1; i <= tot; i++)
49. fa[i] = i;
50. memset(vis, 0, **sizeof** vis);
51. memset(pre, 0, **sizeof** pre);
52. **while**(!que.empty())
53. que.pop();
54. vis[s] = 1;
55. que.push(s);
56. **while**(!que.empty())
57. {
58. **int** nw = que.front();
59. que.pop();
60. trav(nxt, adj[nw])
61. {
62. **if**(find(nw) == find(nxt) || vis[nxt] == 2)
63. **continue**;
64. **if**(!vis[nxt])
65. {
66. vis[nxt] = 2;
67. pre[nxt] = nw;
68. **if**(!match[nxt])
69. {
70. **for**(**int** x = nxt, y; x; x = y)
71. {
72. y = match[pre[x]];
73. match[x] = pre[x];
74. match[pre[x]] = x;
75. }
76. **return** 1;
77. }
78. vis[match[nxt]] = 1, que.push(match[nxt]);
79. }
80. **else**
81. {
82. **int** lca = calc\_lca(nw, nxt);
83. shrink(nw, nxt, lca);
84. shrink(nxt, nw, lca);
85. }
86. }
87. }
88. **return** 0;
89. }
91. **void** sol()
92. {
93. // link edges!
94. **for**(**int** i = 1; i <= tot; i++)
95. {
96. **if**(!match[i])
97. aug(i);
98. }
99. //maybe output by match[]
100. }

KM算法，二分图最大权匹配

1. **int** love[MAXN][MAXN];   // 记录每个妹子和每个男生的好感度
2. **int** ex\_girl[MAXN];      // 每个妹子的期望值
3. **int** ex\_boy[MAXN];       // 每个男生的期望值
4. **bool** vis\_girl[MAXN];    // 记录每一轮匹配匹配过的女生
5. **bool** vis\_boy[MAXN];     // 记录每一轮匹配匹配过的男生
6. **int** match[MAXN];        // 记录每个男生匹配到的妹子 如果没有则为-1
7. **int** slack[MAXN];        // 记录每个汉子如果能被妹子倾心最少还需要多少期望值
8. **int** N;
9. **bool** dfs(**int** girl)
10. {
11. vis\_girl[girl] = 1;
12. **for** (**int** boy = 0; boy < N; ++boy) {
14. **if** (vis\_boy[boy]) **continue**; // 每一轮匹配 每个男生只尝试一次
16. **int** gap = ex\_girl[girl] + ex\_boy[boy] - love[girl][boy];
18. **if** (gap == 0) {  // 如果符合要求
19. vis\_boy[boy] = 1;
20. **if** (match[boy] == -1 || dfs( match[boy] )) {    // 找到一个没有匹配的男生 或者该男生的妹子可以找到其他人
21. match[boy] = girl;
22. **return** 1;
23. }
24. } **else** {
25. slack[boy] = min(slack[boy], gap);  // slack 可以理解为该男生要得到女生的倾心 还需多少期望值 取最小值 备胎的样子【捂脸
26. }
27. }
28. **return** 0;
29. }
30. **int** KM()
31. {
32. memset(match, -1, **sizeof** match);    // 初始每个男生都没有匹配的女生
33. memset(ex\_boy, 0, **sizeof** ex\_boy);   // 初始每个男生的期望值为0
34. // 每个女生的初始期望值是与她相连的男生最大的好感度
35. **for** (**int** i = 0; i < N; ++i) {
36. ex\_girl[i] = love[i][0];
37. **for** (**int** j = 1; j < N; ++j) {
38. ex\_girl[i] = max(ex\_girl[i], love[i][j]);
39. }
40. }
41. // 尝试为每一个女生解决归宿问题
42. **for** (**int** i = 0; i < N; ++i) {
44. fill(slack, slack + N, INF);    // 因为要取最小值 初始化为无穷大
46. **while** (1) {
47. // 为每个女生解决归宿问题的方法是 ：如果找不到就降低期望值，直到找到为止
49. // 记录每轮匹配中男生女生是否被尝试匹配过
50. memset(vis\_girl, **false**, **sizeof** vis\_girl);
51. memset(vis\_boy, **false**, **sizeof** vis\_boy);
53. **if** (dfs(i)) **break**;  // 找到归宿 退出
55. // 如果不能找到 就降低期望值
56. // 最小可降低的期望值
57. **int** d = INF;
58. **for** (**int** j = 0; j < N; ++j)
59. **if** (!vis\_boy[j]) d = min(d, slack[j]);
61. **for** (**int** j = 0; j < N; ++j) {
62. // 所有访问过的女生降低期望值
63. **if** (vis\_girl[j]) ex\_girl[j] -= d;
65. // 所有访问过的男生增加期望值
66. **if** (vis\_boy[j]) ex\_boy[j] += d;
67. // 没有访问过的boy 因为girl们的期望值降低，距离得到女生倾心又进了一步！
68. **else** slack[j] -= d;
69. }
70. }
71. }
72. // 匹配完成 求出所有配对的好感度的和
73. **int** res = 0;
74. **for** (**int** i = 0; i < N; ++i)
75. res += love[ match[i] ][i];
76. **return** res;
77. }

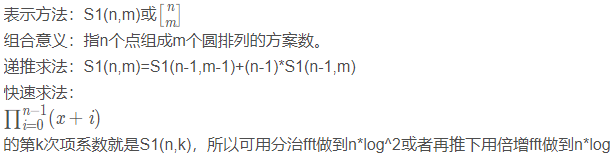
# 八、计数问题&反演容斥&组合数学

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)

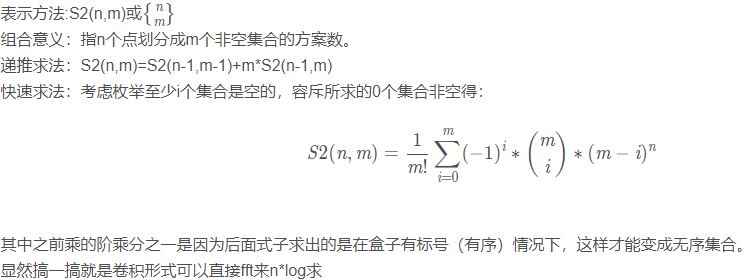
思路：看成走折线图，每次执行一次进栈或出栈操作横坐标加1，同时若进栈纵坐标加1、出栈纵坐标-1，问题等价于从(0,0)到(2n,0)且不碰到y=-1直线方案数。假设某个方案碰到了-1，我们考察该方案从原点第一次碰到-1的走法，把每一步都取反(向上向下互换)，发现可以等价于从(0,-2)碰到-1，那么可能会碰到-1的方案数等价于从(0,-2)走到（2n,0）的方案数，即C(2n, n-1)，故最终方案数C(2n,n) - C(2n,n-1)

2.斯特林数：

（1）第一类：



（2）第二类：



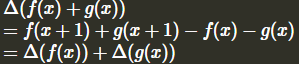
（3）离散微积分：（下降幂重要应用，普通幂则可用斯特林数转化，见（4））



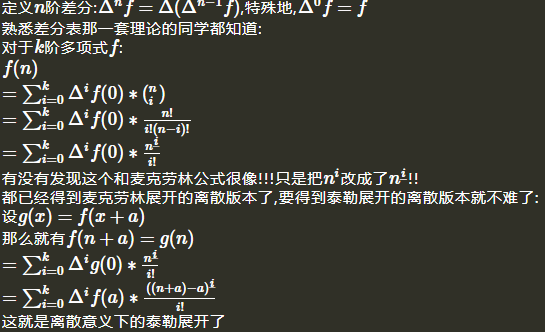




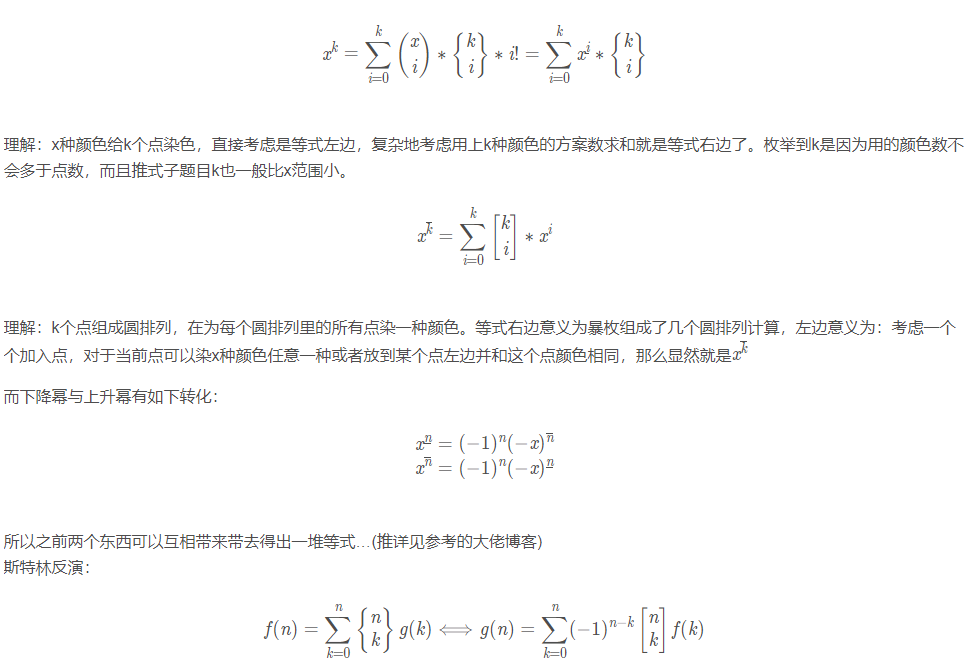




 （）

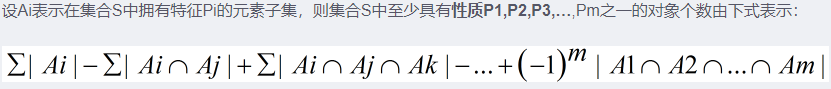


（4）常用斯特林数公式：

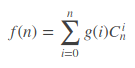


3.容斥：

(1).最经典的排斥包含原理：



(2).二项式反演；

 可以推出： 

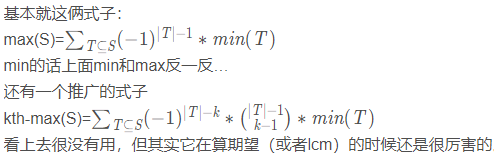
(3).莫比乌斯反演：

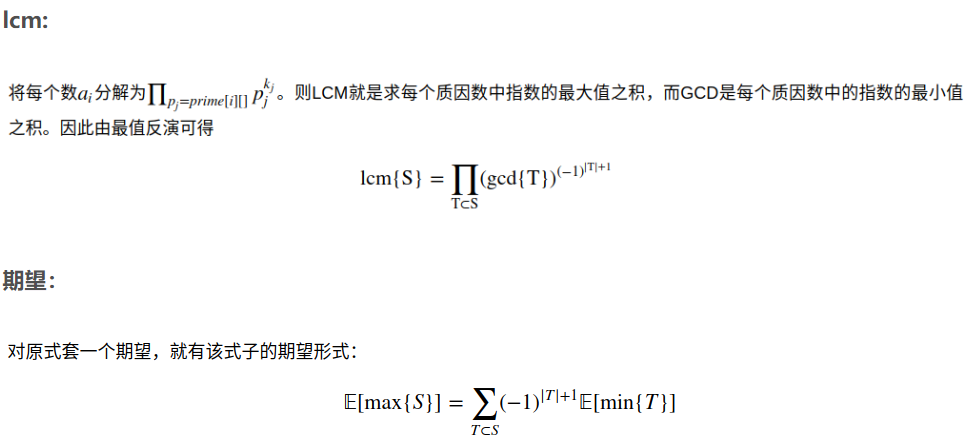
 可以推出：

(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）。

4.递推逆元：



实际上直接ifac[i] \* fac[i – 1]即可

5.组合数前缀和

（1）第一种形式：（证明,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，

1. **const** **int** N = 1e6 + 100;
2. **int** lst[N];
3. **int** pri[N], pnum = 0;
4. **void** init() {
5. **for** (**int** i = 2; i < N; i++) {
6. **if** (!lst[i]) pri[++pnum] = i;
7. **for** (**int** j = 1; j <= pnum && 1LL \* i \* pri[j] < N; j++) {
8. lst[i \* pri[j]] = pri[j];
9. **if** (i % pri[j] == 0) **break**;
10. }
11. }
12. }

值域大时，暴力枚举：

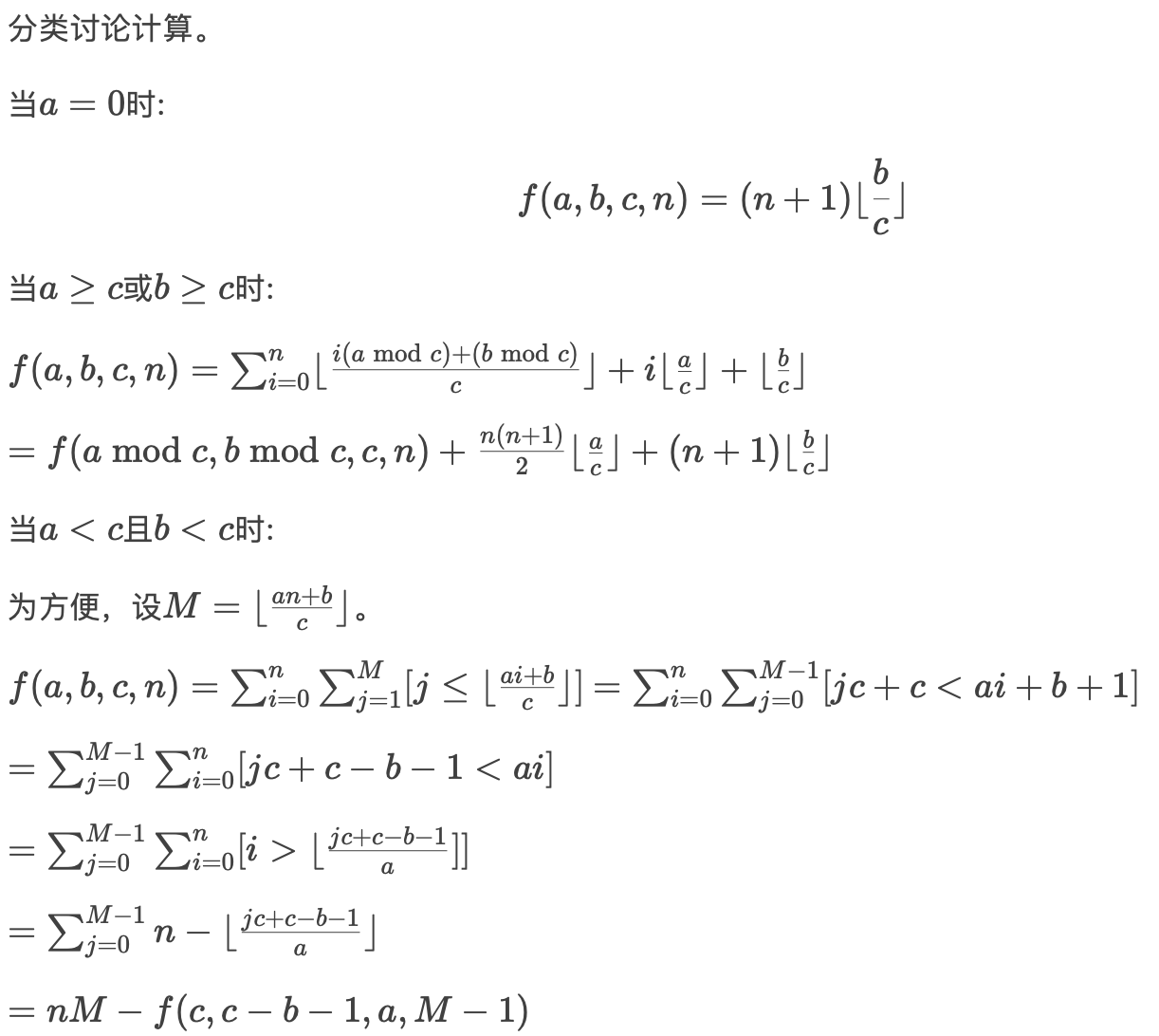
1. **namespace** dec
2. {
3. ll pri[N];
4. **int** pnum;
5. **bool** np[N];
6. **void** init()
7. {
8. **int** n = 1e6;
9. **for**(**int** i = 2; i <= n; i++)
10. {
11. **if**(!np[i])
12. pri[++pnum] = i;
13. **for**(**int** j = 1; j <= pnum && i \* pri[j] <= n; j++)
14. {
15. np[i \* pri[j]] = 1;
16. **if**(i % pri[j] == 0)
17. **break**;
18. }
19. }
20. }
22. ll val[40], cnt[40], cc, cur;
23. VLL buk;
24. **void** dfs(**int** x)
25. {
26. **if**(x > cc)
27. {
28. //cerr << cur << '\n';
29. buk.pb(cur);
30. **return**;
31. }
32. dfs(x + 1);
33. **int** bf = cur;
34. **for**(**int** i = 1; i <= cnt[x]; i++)
35. {
36. cur \*= val[x];
37. dfs(x + 1);
38. }
39. cur = bf;
40. }
41. VLL work(ll nw)
42. {
43. cc = 0, buk.clear();
44. **for**(**int** i = 1; i <= pnum && pri[i] <= nw / pri[i]; i++)
45. {
46. **if**(nw % pri[i])
47. **continue**;
48. val[++cc] = pri[i];
49. cnt[cc] = 0;
50. **while**(nw % pri[i] == 0)
51. ++cnt[cc], nw /= pri[i];
52. }
53. **if**(nw > 1)
54. val[++cc] = nw, cnt[cc] = 1;
55. cur = 1, dfs(1);
56. sort(all(buk));
57. **return** buk;
58. }
59. }

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. }

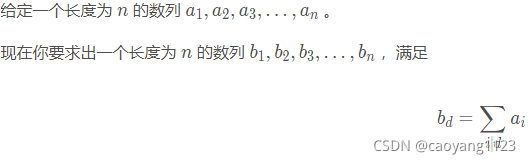
2.5类欧几里得算法

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. }

3.迪利克雷前后缀和：



发现a到b有贡献，当且仅当a中每个质因子出现幂次小于等于b中对应质因子出现幂次。

然后发现相当于对每个质因子幂次从低到高做前缀和，可行的从a到b就是先走小质因子再走大：

如6贡献到36：

枚举到质因子2:6->12->24->... 枚举到质因子3:12->36...

发现这样6就贡献到36了

故可以枚举质因子，然后暴力枚举1~n，复杂度sigma （n/pi）,为nloglog，与埃氏筛相同。

1. **for**(**int** i = 1; i <= cnt && primes[i] <= n; ++ i) {
2. **for**(**int** j = 1; j \* primes[i] <= n; ++ j) {
3. a[j \* primes[i]] += a[j];
4. }
5. }



1. **for**(**int** i = 1; i <= cnt && primes[i] <= n; ++ i) {
2. **for**(**int** j = n / primes[i]; j ; -- j) {
3. a[j] += a[j \* primes[i]];
4. }
5. }

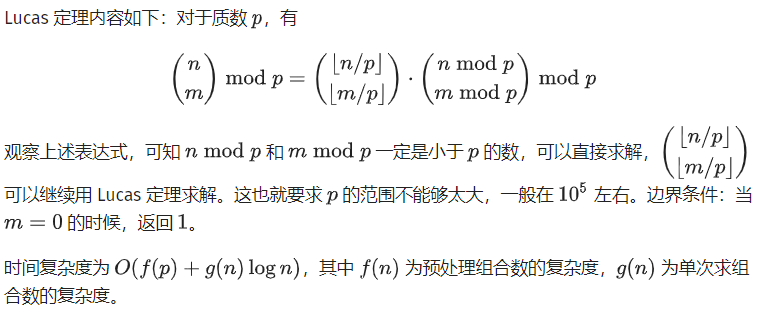
倒推前缀和，由b到a：

1. **for**(**int** i = cnt; i ; -- i) {
2. **for**(**int** j = n / primes[i]; j ; -- j) {
3. a[j \* primes[i]] -= a[j];
4. }
5. }

倒推后缀和：

1. **for**(**int** i = cnt; i ; -- i) {
2. **for**(**int** j = 1; j \* primes[i] <= n; ++ j) {
3. a[j] -= a[j \* primes[i]];
4. }
5. }

4.卢卡斯定理



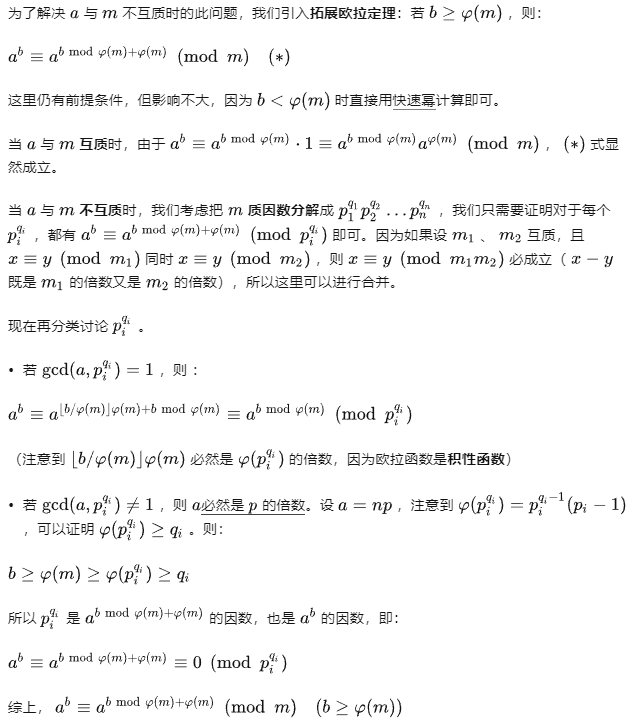
5.皮克定理：

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，证明：证矩形以及直角三角形满足，再证图形相加也满足，然后任何一个三角形都可以分解为一个矩形减去三个直角三角形，任何一个多边形都可以分解为三角形相加.

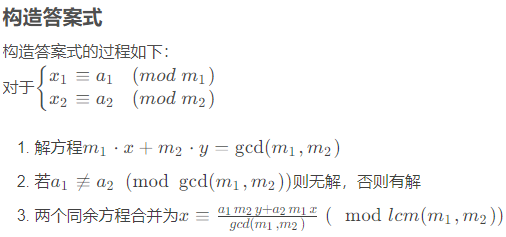
应用，对面积和内部格点的要求可以转化为对边界格点要求，然后运用同余知识。

6.欧拉定理&扩展欧拉定理

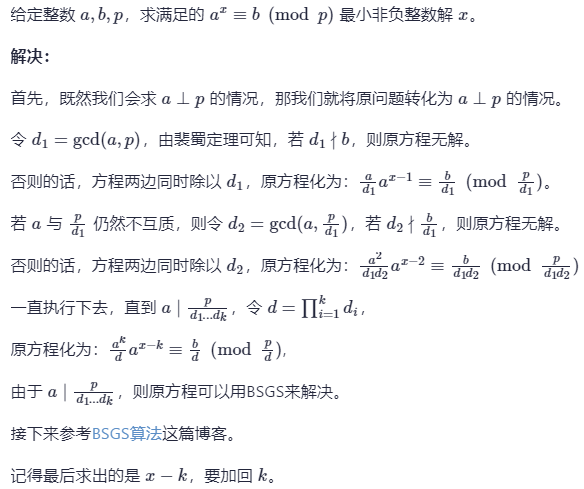
**欧拉定理：  
扩展欧拉定理：**



7.扩展中国剩余定理excrt

****

8.扩展拔山盖世EXBSGS

****

1. LL exBSGS(LL a,LL b,LL p){
2. a%=p,b%=p;
3. **if**(b==1||p==1)**return** 0;
4. LL d,k=0,A=1;
5. **while**(1){
6. d=gcd(a,p);
7. **if**(d==1)**break**;
8. **if**(b%d!=0)**return** -1;
9. k++,b/=d,p/=d;
10. A=A\*(a/d)%p;
11. **if**(A==b)**return** k;
12. }
13. LL m=ceil(sqrt(p)),t=b;
14. unordered\_map<**int**,**int**>Hash;
15. Hash[b]=0;
16. **for**(**int** j=1;j<m;j++){
17. t=t\*a%p;
18. Hash[t]=j;
19. }
20. LL mi=1;
21. **for**(**int** i=1;i<=m;i++)mi=mi\*a%p;
22. t=A;
23. **for**(**int** i=1;i<=m;i++){
24. t=t\*mi%p;
25. **if**(Hash.count(t))**return** i\*m-Hash[t]+k;
26. }
27. **return** -1;
28. }

# 十、数据结构

-1.0.pbds （内置hash表&可查排名的红黑树）

1. #include<ext/pb\_ds/assoc\_container.hpp>
2. #include<ext/pb\_ds/tree\_policy.hpp>
3. #include<ext/pb\_ds/hash\_policy.hpp>
4. **using** **namespace** std;
5. **using** **namespace**  \_\_gnu\_pbds;

8. gp\_hash\_table<string, **int**> h;
9. /\*
10. support [] and find and erase (should directly erase key without find)
11. and count (must be 0 or 1) and clear and size and empty
12. \*/
14. tree<pii, null\_type, less<pii>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> rb\_tree;
15. /\*
16. !!!!!!!!!! not support same element !!!!!!!!!!
17. .
18. pii: type
19. null\_type
20. less<pii>: sorted in increasing order
21. rb\_tree\_tag: rb\_tree
22. tree\_order\_statistics\_node\_update: update method
23. .insert(pii()), .erase(pii())
24. .order\_of\_key(pii()) find rnk = #strictly small than x
25. .find\_by\_order(k) number k less value, start from 0
26. .split(v, b) less or equal v belong to self, and others belong to b
27. .join(b) merge self and b
28. .lower\_bound(), .upper\_bound return iterator
29. \*/
30. **void** sol()
31. {
32. **int** qq;
33. cin >> qq;
34. **while** (qq--) {
35. **int** t, x;
36. cin >> t >> x;
37. **if** (t == 1) {
38. pii nw = pii(x, ++cnt[x]);
39. rb\_tree.insert(nw);
40. } **else** **if** (t == 2) {
41. pii nw = pii(x, cnt[x]);
42. rb\_tree.erase(nw);
43. --cnt[x];
44. } **else** **if** (t == 3) {
45. **int** rnk = rb\_tree.order\_of\_key(pii(x, 0)) + 1;
46. cout << rnk << '\n';
47. } **else** **if** (t == 4) {
48. auto res = \*rb\_tree.find\_by\_order(x - 1);
49. cout << res.fi << '\n';
50. } **else** **if** (t == 5) {
51. auto res = \*prev(rb\_tree.lower\_bound(pii(x, 0)));
52. cout << res.fi << '\n';
53. } **else** {
54. auto res = \*(rb\_tree.upper\_bound(pii(x, 1e9)));
55. cout << res.fi << '\n';
56. }
57. }
58. }

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
129. (2) implement struct Lazy
130. (3) implement Info operator +(Info a, Info b)
131. (4) implement Info operator +(Info a, Lazy b)
132. (5) implement Lazy operator +(Lazy a, Lazy b)
134. \*when not need Lazy, use SegTree<Info, Info>, so can only implement (1) (3)
135. \*Lazy can be long long or some default type, so can only implement (1) (3) (4)
137. .init(vector bg/size\_bg) should give vector or size
138. vector first element should start from 1 (index 0 not use)
139. \*/

1/4.并查集

1. **struct** DSU {
2. vector<**int**> fa;
3. vector<**int**> sz;
4. DSU() {}
5. DSU(**int** n) {
6. init(n);
7. }
8. **void** init(**int** n) {
9. fa.resize(n + 5);
10. iota(all(fa), 0);
11. sz.resize(n + 5);
12. fill(all(sz), 1);
13. sz[0] = 0;
14. }
15. **int** find(**int** x) {
16. **return** x == fa[x] ? x : fa[x] = find(fa[x]);
17. }
18. **bool** same(**int** x, **int** y) {
19. **return** find(x) == find(y);
20. }
21. **int** merge(**int** x, **int** y) {
22. **int** fx = find(x);
23. **int** fy = find(y);
24. **if** (fx == fy) **return** 0;
25. **if** (sz[fx] > sz[fy]) {
26. swap(x, y);
27. swap(fx, fy);
28. }
29. fa[fx] = fy;
30. sz[fy] += sz[fx];
31. **return** 1;
32. }
33. **int** size(**int** x) {
34. **return** sz[find(x)];
35. }
36. };

1/2.Hash表

1. **const** **int** Hash\_mod = 1e7 + 100;
2. **struct** Hash\_table {
3. vector<**int**> buk[Hash\_mod];
4. **void** ins(**int** x) {
5. buk[x % Hash\_mod].pb(x);
6. }
7. **int** find(**int** x) {
8. **int** wh = x % Hash\_mod;
9. **for** (**int** i = buk[wh].size() - 1; i >= 0; i--) {
10. **if** (buk[wh][i] == x) {
11. **return** i;
12. }
13. }
14. **return** -1;
15. }
16. **void** del(**int** x, **int** i) {
17. **int** wh = x % Hash\_mod;
18. **if** (i == buk[wh].size() - 1) buk[wh].pop\_back();
19. **else** buk[wh].erase(buk[wh].begin() + i);
20. }
21. }hsh\_T;

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. }

4.LCT（注意修改操作一般都要，旋转到两点在一个splay里，即在一个splay维护的一段路径中，使用split操作！！）

1. **namespace** LCT{
2. #define ls ch[0]
3. #define rs ch[1]
4. **int** tot;
5. **struct** gg{
6. **int** ch[2], fa, las, val, mx, ps;
7. **bool** rev;
8. **void** in(**int** x, **int** y)
9. {
10. las = x, val = y, ps = tot;
11. ch[0] = ch[1] = fa = mx = 0;
12. rev = 0;
13. }
14. }node[N];
15. **void** init()
16. {
17. **for**(**int** i = 0; i <= tot; i++)
18. {
19. node[i].ch[0] = node[i].ch[1] = node[i].fa = node[i].rev = 0;
20. node[i].val = node[i].las = node[i].mx = 0;
21. }
22. tot = 0;
23. }//毫无卵用的清空
24. **int** get(**int** x)//询问该点是父亲的左儿子或右儿子或都不是（都不是即该点为所在splay的根，通过轻边单向连父亲）
25. {
26. **if**(!node[x].fa)
27. **return** -1;
28. **if**(node[node[x].fa].ch[0]==x)
29. **return** 0;
30. **if**(node[node[x].fa].ch[1]==x)
31. **return** 1;
32. **return** -1;
33. }
34. **void** push\_up(**int** x)
35. {
36. node[x].mx = node[x].val, node[x].ps = x;
37. **int** l = node[x].ls, r = node[x].rs;
38. **if**(l && node[x].mx < node[l].mx)
39. node[x].mx = node[l].mx, node[x].ps = node[l].ps;
40. **if**(l && node[x].mx == node[l].mx && node[x].ps < node[l].ps)
41. node[x].mx = node[l].mx, node[x].ps = node[l].ps;
42. swap(l, r);
43. **if**(l && node[x].mx < node[l].mx)
44. node[x].mx = node[l].mx, node[x].ps = node[l].ps;
45. **if**(l && node[x].mx == node[l].mx && node[x].ps < node[l].ps)
46. node[x].mx = node[l].mx, node[x].ps = node[l].ps;
47. }
48. **void** push\_down(**int** x)//下放翻转标记，同普通splay
49. {
50. **if**(x && node[x].rev)
51. {
52. node[node[x].ch[0]].rev^=1;
53. node[node[x].ch[1]].rev^=1;
54. swap(node[x].ch[0],node[x].ch[1]);
55. node[x].rev=0;
56. }
57. }
58. **void** push(**int** x)//下放x点到所在splay的根的所有标记
59. {
60. **if**(get(x) != -1)push(node[x].fa);
61. push\_down(x);
62. }
63. **void** rotate(**int** x)//splay基本操作，注意判断父亲是所在splay根的情况
64. {
65. **int** fx = node[x].fa, gx = node[fx].fa, op = get(x), fop = get(fx);
66. node[fx].ch[op] = node[x].ch[op ^ 1], node[node[x].ch[op ^ 1]].fa = fx;
67. node[x].fa = gx;
68. **if**(fop!=-1)
69. node[gx].ch[fop] = x;
70. node[x].ch[op ^ 1] = fx, node[fx].fa = x;
71. push\_up(fx);
72. push\_up(x);
73. }
74. **void** splay(**int** x)//splay基本操作，注意先下放标记
75. {
76. push(x);
77. **for**(**int** fx = node[x].fa; get(x) != -1; rotate(x), fx = node[x].fa)
78. {
79. **if**(get(fx)!=-1)
80. {
81. (get(x)==get(fx))?rotate(fx):rotate(x);
82. }
83. }
84. }
85. **void** access(**int** x)//最重要操作，即我在前一篇说的更改轻重链，它能将x节点与整棵树根（不是所在splay）连到一个重链里
86. //并且x节点和儿子都以虚边连接以支持换根
87. {
88. **for**(**int** y = 0; x;)
89. {
90. splay(x);//先转到当前splay的根
91. node[x].ch[1] = y;//右儿子断开，变为上次转到splay的点
92. push\_up(x);
93. y = x, x = node[x].fa;//更新
94. }
95. }
96. **void** make\_root(**int** x)//换根操作，令x成为整棵树根（基于access后x为所在splay最深点性质）
97. {
98. access(x), splay(x), node[x].rev ^= 1;//打标记是因为x变为根后它的深度要变为最小，因此需翻转
99. }
100. **void** split(**int** x, **int** y)//把x与y的路径组成一个以y为根的splay
101. {
102. make\_root(x), access(y), splay(y);
103. }
104. **void** cut(**int** x,**int** y)//把x与y分离
105. {
106. split(x,y);
107. node[x].fa = node[y].ch[0] = 0;
108. }
109. **void** line(**int** x,**int** y)//连接x，y
110. {
111. node[x].fa=y;
112. }
113. **int** find(**int** x)//查找x点所在树的根节点
114. {
115. **for**(access(x), splay(x); node[x].ch[0]; x = node[x].ch[0]);
116. **return** x;
117. }
118. **int** find\_ps(**int** x)
119. {
120. split(x, 1);
121. **if**(node[1].mx <= 0)
122. **return** -1;
123. **return** node[1].ps;
124. }
125. }
126. **using** **namespace** LCT;
128. **int** idx[N];
129. **int** qq;
131. pll ask(**int** x, **int** y)
132. {
133. ll res = 0, num = 0;
134. **while**(y)
135. {
136. **int** ps = find\_ps(x);
137. //cerr << ps << ' ' << node[ps].val << ' ' << node[ps].las << '\n';
138. **if**(ps < 0)
139. **break**;
140. **int** tmp = min(node[ps].las, y);
141. node[ps].las -= tmp;
142. y -= tmp;
143. num += tmp;
144. res += 1LL \* node[ps].val \* tmp;
145. **if**(node[ps].las == 0)
146. {
147. make\_root(ps);
148. node[ps].las = 0;
149. node[ps].val = -1;
150. node[ps].mx = -1;
151. }
152. }
153. **return** pll(res, num);
154. }

5.K-D tree

维护平面上的点，1、2维轮流使用找中间元素进行划分，构成的一颗二叉搜索树就是KD-tree，插入点后容易导致不平衡，可以类似替罪羊树拍扁重建

关于确定好每一层按哪个关键字后，我们就把点按照关键字排序，并取当前数列中的中间那一个，再对它的下一层进行遍历

1. **struct** gg{
2. **int** ls,rs,l,r,up,dw,d[2];
3. }node[N];
4. **int** x[N],y[N],n,D,rt,ans1,ans2,X,Y;
5. **bool** cmp(gg a,gg b)
6. {**return** a.d[D]==b.d[D]?a.d[D^1]<b.d[D^1]:a.d[D]<b.d[D];}
7. **void** push\_up(**int** x,**int** y)
8. {
9. node[x].dw=min(node[x].dw,node[y].dw);
10. node[x].up=max(node[x].up,node[y].up);
11. node[x].l=min(node[x].l,node[y].l);
12. node[x].r=max(node[x].r,node[y].r);
13. }
14. **void** build(**int** l,**int** r,**int** &nw,**int** dd)
15. {
16. D=dd;**int** mid=(l+r)>>1;
17. nth\_element(node+l,node+mid,node+r+1,cmp);
18. nw=mid;
19. node[nw].dw=node[nw].up=node[nw].d[1];
20. node[nw].l=node[nw].r=node[nw].d[0];
21. **if**(l<mid)build(l,mid-1,node[nw].ls,dd^1),push\_up(nw,node[nw].ls);**else** node[nw].ls=0;
22. **if**(r>mid)build(mid+1,r,node[nw].rs,dd^1),push\_up(nw,node[nw].rs);**else** node[nw].rs=0;
23. }
24. **int** dis1(**int** x)
25. {
26. **int** res=0;
27. res=max(abs(node[x].up-Y),abs(node[x].dw-Y))+max(abs(node[x].l-X),abs(node[x].r-X));
28. **return** res;
29. }
30. **int** dis2(**int** x)
31. {
32. **int** res=0;
33. **if**(node[x].dw>Y)res+=node[x].dw-Y;
34. **if**(node[x].up<Y)res+=Y-node[x].up;
35. **if**(node[x].l>X)res+=node[x].l-X;
36. **if**(node[x].r<X)res+=X-node[x].r;
37. **return** res;
38. }
39. **void** ask1(**int** x)
40. {
41. ans1=max(ans1,abs(node[x].d[0]-X)+abs(node[x].d[1]-Y));
42. **int** dl=-2e9,dr=-2e9;
43. **if**(node[x].ls)dl=dis1(node[x].ls);
44. **if**(node[x].rs)dr=dis1(node[x].rs);
45. **if**(dl>dr)
46. {
47. **if**(dl>ans1)ask1(node[x].ls);
48. **if**(dr>ans1)ask1(node[x].rs);
49. }
50. **else**
51. {
52. **if**(dr>ans1)ask1(node[x].rs);
53. **if**(dl>ans1)ask1(node[x].ls);
54. }
55. }
56. **void** ask2(**int** x)
57. {
58. **int** nw=abs(node[x].d[0]-X)+abs(node[x].d[1]-Y);
59. **if**(nw)ans2=min(ans2,nw);
60. **int** dl=2e9,dr=2e9;
61. **if**(node[x].ls)dl=dis2(node[x].ls);
62. **if**(node[x].rs)dr=dis2(node[x].rs);
63. **if**(dl<dr)
64. {
65. **if**(dl<ans2)ask2(node[x].ls);
66. **if**(dr<ans2)ask2(node[x].rs);
67. }
68. **else**
69. {
70. **if**(dr<ans2)ask2(node[x].rs);
71. **if**(dl<ans2)ask2(node[x].ls);
72. }
73. }
74. **bool** bad(**int** k)//如替罪羊树一样判断是否平衡，alpha=0.725
75. {
76. **return** (t[k].siz\*alpha<t[t[k].lc].siz||t[k].siz\*alpha<t[t[k].rc].siz);
77. }
78. **void** work(**int** k)
79. {
80. **if**(t[k].lc)work(t[k].lc);
81. p[++cnt]=t[k].pt;
82. rub[++topp]=k;//将不用的节点编号存进rub中，节省空间
83. **if**(t[k].rc)work(t[k].rc);
84. }
85. **int** build(**int** l,**int** r,**int** wd)
86. {
87. **if**(l>r)**return** 0;
88. **int** mid=(l+r)>>1,k=newnode();
89. WD=wd;//每次按照当前维度排序
90. nth\_element(p+l,p+mid,p+r+1,cmp);
91. //这是一个神奇的STL，会使得序列a中[l,r]中的第mid小的元素在第mid位上，但是其他元素并不有序！！！
92. //这个STL的时间复杂度为O(n)，这也是我们不使用sort的原因，并且可以去到中位数
93. t[k].pt=p[mid];
94. t[k].lc=build(l,mid-1,wd^1);
95. t[k].rc=build(mid+1,r,wd^1);
96. update(k);
97. **return** k;
98. }
99. **void** rebuild(**int** &k)
100. {
101. cnt=0;
102. work(k);//拍扁
103. k=build(1,cnt,0);//重建
104. }

6.带修改主席树（单点修改，区间第k小）：在主席树的外层套树状数组，即下标为x的线段树维护x～x+lowbit(x)的区间，修改很简单，查询的话就把log个需要用的树提取出来，每一层查询个数和k比，在树上往左往右走。

1. **struct** opt{
2. **bool** op;
3. **int** l,r,k;
4. opt(){};
5. opt(**bool** op,**int** l,**int** r,**int** k):op(op),l(l),r(r),k(k){};
6. }a[N];
8. **struct** Seg{
9. **int** ls,rs,sz;
10. }seg[N\*400];**int** snum;
12. **int** n,m,las[N],rt[N],tax[N],tnum,q1[N],q2[N],qnum1=0,qnum2=0;
14. **void** lisan()
15. {
16. sort(tax+1,tax+tnum+1);
17. tnum=unique(tax+1,tax+tnum+1)-tax-1;
18. **for**(**int** i=1;i<=n;i++)las[i]=lower\_bound(tax+1,tax+tnum+1,las[i])-tax;
19. **for**(**int** i=1;i<=m;i++)
20. **if**(a[i].op==1)a[i].r=lower\_bound(tax+1,tax+tnum+1,a[i].r)-tax;
21. }
23. **int** ins(**int** bf,**int** l,**int** r,**int** to,**int** val)
24. {
25. **int** nw=++snum;
26. seg[nw]=seg[bf];
27. seg[nw].sz+=val;
28. **if**(l!=r)
29. {
30. **int** mid=(l+r)>>1;
31. **if**(to<=mid)seg[nw].ls=ins(seg[bf].ls,l,mid,to,val);
32. **else** seg[nw].rs=ins(seg[bf].rs,mid+1,r,to,val);
33. }
34. **return** nw;
35. }
37. **void** add(**int** x,**int** y,**int** val)
38. {**for**(;x<=n;x+=x&-x)rt[x]=ins(rt[x],1,tnum,y,val);}
40. **int** calc()
41. {
42. **int** res=0;
43. **for**(**int** i=1;i<=qnum1;i++)res-=seg[seg[q1[i]].ls].sz;
44. **for**(**int** i=1;i<=qnum2;i++)res+=seg[seg[q2[i]].ls].sz;
45. **return** res;
46. }
48. **int** qry(**int** l,**int** r,**int** k)
49. {
50. qnum1=qnum2=0;
51. **for**(**int** i=l-1;i;i-=i&-i)q1[++qnum1]=rt[i];
52. **for**(**int** i=r;i;i-=i&-i)q2[++qnum2]=rt[i];
53. **int** lq=1,rq=tnum;
54. **while**(lq<rq)
55. {
56. **int** mid=(lq+rq)>>1,cnt=calc();
57. **if**(cnt>=k)
58. {
59. **for**(**int** i=1;i<=qnum1;i++)q1[i]=seg[q1[i]].ls;
60. **for**(**int** i=1;i<=qnum2;i++)q2[i]=seg[q2[i]].ls;
61. rq=mid;
62. }
63. **else**
64. {
65. **for**(**int** i=1;i<=qnum1;i++)q1[i]=seg[q1[i]].rs;
66. **for**(**int** i=1;i<=qnum2;i++)q2[i]=seg[q2[i]].rs;
67. k-=cnt,lq=mid+1;
68. }
69. }
70. **return** lq;
71. }
73. **void** sol()
74. {
75. **for**(**int** i=1;i<=n;i++)add(i,las[i],1);
76. **for**(**int** i=1;i<=m;i++)
77. {
78. **if**(a[i].op)
79. {
80. add(a[i].l,las[a[i].l],-1);
81. las[a[i].l]=a[i].r;
82. add(a[i].l,las[a[i].l],1);
83. }
84. **else** printf("%d\n",tax[qry(a[i].l,a[i].r,a[i].k)]);
85. }
86. }

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,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. }

9.zkw线段树

1. **void** build()
2. {
3. **int** N = 1 << \_\_lg(n + 5) + 1;
4. vector<**int**> tree(N << 1);
5. **for** (**int** i = 1; i <= n ++i)
6. tree[i + N] = A[i];
7. **for** (**int** i = N; i; --i)
8. tree[i] = tree[i << 1] + tree[i << 1 | 1];
9. }
10. **void** update(**int** x, **int** d) // 单点修改
11. {
12. **for** (**int** i = x + N; i; i >>= 1)
13. tree[i] += d;
14. }
15. **int** query(**int** l, **int** r) // 区间查询
16. {
17. **int** ans = 0;
18. **for** (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1)
19. {
20. **if** (~l & 1) ans += tree[l ^ 1]; // 左端点是左儿子
21. **if** (r & 1) ans += tree[r ^ 1]; // 右端点是右儿子
22. }
23. **return** ans;
24. }
25. **void** update(**int** l, **int** r, **int** d) // 区间修改
26. {
27. **int** len = 1, cntl = 0, cntr = 0; // cntl、cntr是左、右两边分别实际修改的区间长度
28. **for** (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1, len <<= 1)
29. {
30. tree[l] += cntl \* d, tree[r] += cntr \* d;
31. **if** (~l & 1) tree[l ^ 1] += d \* len, mark[l ^ 1] += d, cntl += len;
32. **if** (r & 1) tree[r ^ 1] += d \* len, mark[r ^ 1] += d, cntr += len;
33. }
34. **for** (; l; l >>= 1, r >>= 1)
35. tree[l] += cntl \* d, tree[r] += cntr \* d;
36. }
37. **int** query(**int** l, **int** r) // 区间查询带标记
38. {
39. **int** ans = 0, len = 1, cntl = 0, cntr = 0;
40. **for** (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1, len <<= 1)
41. {
42. ans += cntl \* mark[l] + cntr \* mark[r];
43. **if** (~l & 1) ans += tree[l ^ 1], cntl += len;
44. **if** (r & 1) ans += tree[r ^ 1], cntr += len;
45. }
46. **for** (; l; l >>= 1, r >>= 1)
47. ans += cntl \* mark[l] + cntr \* mark[r];
48. **return** ans;
49. }
50. // 区间加，查询区间最大值
51. **void** update(**int** l, **int** r, **int** d) {
52. **for** (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1)
53. {
54. **if** (l < N) tree[l] = max(tree[l << 1], tree[l << 1 | 1]) + mark[l],
55. tree[r] = max(tree[r << 1], tree[r << 1 | 1]) + mark[r];
56. **if** (~l & 1) tree[l ^ 1] += d, mark[l ^ 1] += d;
57. **if** (r & 1) tree[r ^ 1] += d, mark[r ^ 1] += d;
58. }
59. **for** (; l; l >>= 1, r >>= 1)
60. **if** (l < N) tree[l] = max(tree[l << 1], tree[l << 1 | 1]) + mark[l],
61. tree[r] = max(tree[r << 1], tree[r << 1 | 1]) + mark[r];
62. };
63. **int** query(**int** l, **int** r) {
64. **int** maxl = -INF, maxr = -INF;
65. **for** (l += N - 1, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1)
66. {
67. maxl += mark[l], maxr += mark[r];
68. **if** (~l & 1) cmax(maxl, tree[l ^ 1]);
69. **if** (r & 1) cmax(maxr, tree[r ^ 1]);
70. }
71. **for** (; l; l >>= 1, r >>= 1)
72. maxl += mark[l], maxr += mark[r];
73. **return** max(maxl, maxr);
74. };

# 十一、图论问题

迪杰斯特拉和最短路树：

最短路树：定义构建一棵树，使得任意不属于根的节点x，dis(root,x)=原图走到x的最短路。

构建方法：就是在跑dijkstra时同时维护每个点是哪个点哪条边更新的，这个点这条边就是它在最短路树上的父亲/到父亲的边

1. **void** add(**int** x,**int** y,**int** z,**int** id)
2. {
3. nxt[++tot]=hd[x],idx[tot]=id,to[tot]=y,cost[tot]=z,hd[x]=tot;
4. nxt[++tot]=hd[y],idx[tot]=id,to[tot]=x,cost[tot]=z,hd[y]=tot;
5. }
7. **void** dij()
8. {
9. priority\_queue<pii,vector<pii>,greater<pii> >pq;
10. **int** nw;ll d;
11. memset(dis,63,**sizeof** dis);
12. dis[1]=0,pq.push(pii(0,1));
13. **while**(!pq.empty())
14. {
15. nw=pq.top().sc,d=pq.top().fi,pq.pop();
16. **if**(dis[nw]<d)**continue**;
17. **for**(**int** i=hd[nw],v;~i;i=nxt[i])
18. {
19. v=to[i];
20. **if**(dis[v]>d+cost[i])
21. {
22. dis[v]=d+cost[i];
23. bf[v]=idx[i];
24. pq.push(pii(dis[v],v));
25. }
26. }
27. }
28. }

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。

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，如果不等式组中有xi – xj > k，因为一般题目都是对整形变量的约束，化为xi – xj >= k+1即可，如果xi – xj = k呢，那么可以变为如下两个：xi – xj >= k, xi – xj <= k,进一步变为xj – xi >= -k，建立两条边即可。

(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，如果像上面的两种情况，那么同样地标准化就行了。

(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，我们只需要加入一个超级源点，比如说求取最长路时图不联通的话，我们只需要加入一个点S，对其他的每个点建立一条权值为0的边图就联通了，然后从S点开始进行spfa判环。最短路类似。

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：

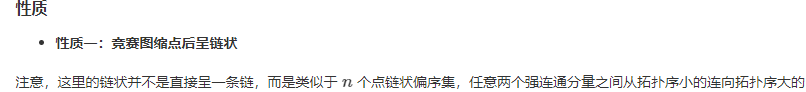
dfs（x）：

若某条边（x，y）未访问

标记为已访问，dfs（y），把y入栈。

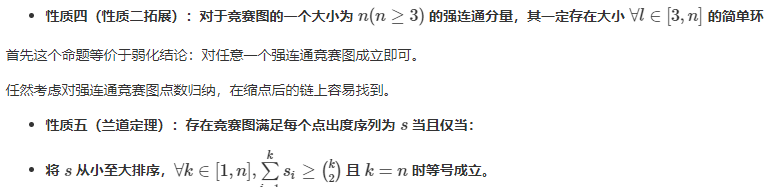
倒序输出为答案

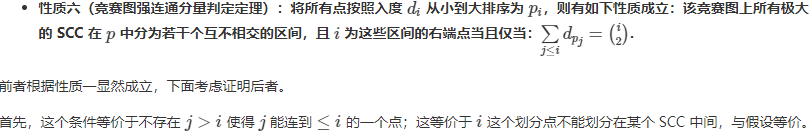
竞赛图：











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）。

有一些没啥用的定理：

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。

求边双：将桥边删除，每个连通块都是一个边双。而将每个边双看成一个点建新图即可完成边双缩点。

求点双：首先对于孤立点（没有连边的）直接自己作为点双，其余的就在tarjan时维护一个栈，一个点第一次访问时加入栈，当出现dfn（x）<=low（y），从栈顶不断弹，弹出y为止，所有弹出的和x（x为割点）构成点双。之后缩点则将每个割点和包含它的点双连边构成森林。（新开一个点对应点双，割点向所在点双连边）

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:

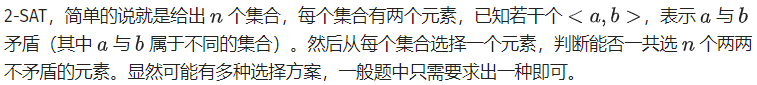
1. **struct** EBCC {
2. **int** n;
3. vector<vector<**int**>> adj;
4. vector<**int**> stk;
5. vector<**int**> dfn, low, bel;
6. **int** tim, cnt;
8. EBCC() {}
9. EBCC(**int** n) {
10. init(n);
11. }
13. **void** init(**int** n) {
14. **this**->n = n;
15. adj.assign(n + 5, {});
16. dfn.assign(n + 5, 0);
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. };

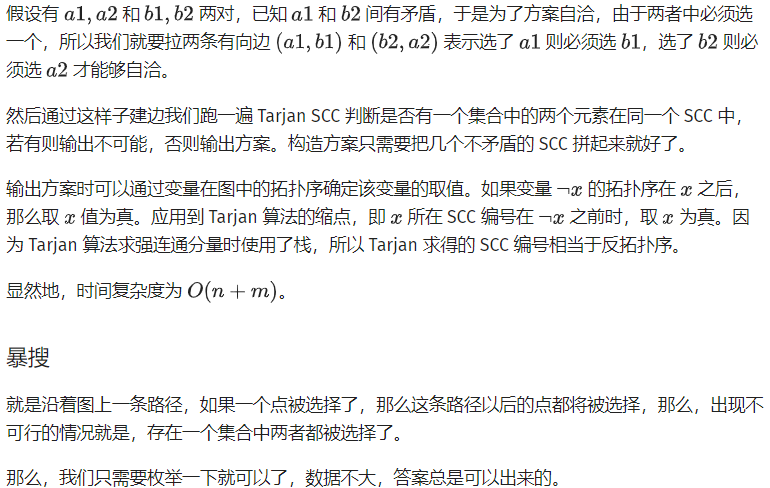
**（2）有向图tarjan**

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）

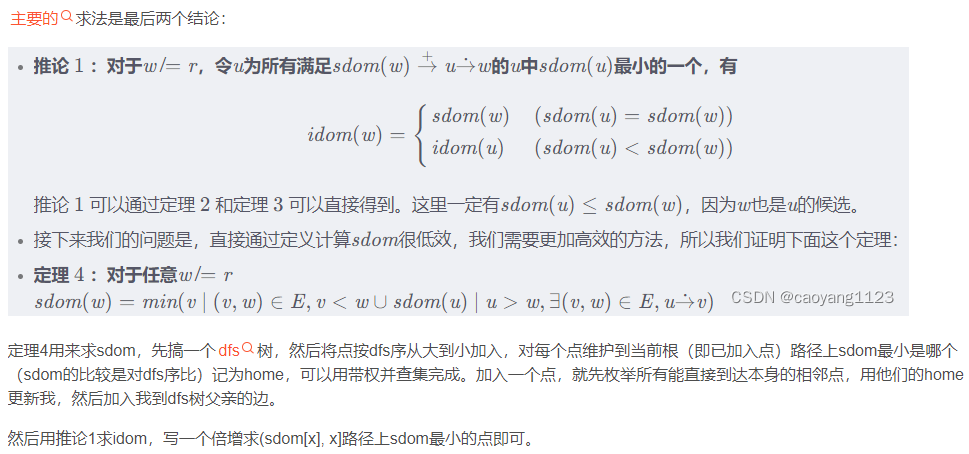
**tarjan法代码**

1. **int** n, m;
3. vector<**int**> adj[N];
5. **int** dfn[N], low[N], tim;
6. **bool** ins[N];
7. vector<**int**> stk;
9. **bool** hav[N];
11. **int** bel[N], num = 0;
13. **void** dfs(**int** x)
14. {
15. dfn[x] = low[x] = ++tim;
16. ins[x] = 1;
17. stk.pb(x);
18. trav(v, adj[x])
19. {
20. **if**(!dfn[v])
21. dfs(v), low[x] = min(low[x], low[v]);
22. **else** **if**(ins[v])
23. low[x] = min(low[x], dfn[v]);
24. }
25. **if**(low[x] == dfn[x])
26. {
27. ++num;
28. VI bin;
29. **while**(1)
30. {
31. **int** nw = stk.back();
32. stk.pop\_back();
33. **if**(nw > n && hav[nw - n])
34. {
35. cout << "IMPOSSIBLE" << '\n';
36. exit(0);
37. }
38. **if**(nw < n && hav[nw + n])
39. {
40. cout << "IMPOSSIBLE" << '\n';
41. exit(0);
42. }
43. hav[nw] = 1;
44. ins[nw] = 0;
45. bel[nw] = num;
46. bin.pb(nw);
47. **if**(nw == x)
48. **break**;
49. }
50. trav(v, bin)
51. hav[v] = 0;
52. bin.clear();
53. }
54. }
56. **void** sol()
57. {
58. cin >> n >> m;
59. **for**(**int** i = 1; i <= m; i++)
60. {
61. **int** x, a, y, b, rx, ry;
62. cin >> x >> a >> y >> b;
63. rx = x + n, ry = y + n;
64. **if**(a == 1)
65. swap(x, rx);
66. **if**(b == 1)
67. swap(y, ry);
68. adj[rx].pb(y);
69. adj[ry].pb(x);
70. }
71. **for**(**int** i = 1; i <= n + n; i++)
72. **if**(!dfn[i])dfs(i);
73. cout << "POSSIBLE" << '\n';
74. **for**(**int** i = 1; i <= n; i++)
75. cout << (bel[i] < bel[i + n] ? 0 : 1) << " \n"[i == n];
76. }

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. };

（4）支配树：

****

1. **void** sol(**int** n, **int** m) {
2. vector<vector<**int**>> adj(n + 1), radj(n + 1), tree\_adj(n + 1);
3. **for** (**int** i = 1; i <= m; i++) {
4. **int** x, y;
5. cin >> x >> y;
6. adj[x].pb(y);
7. radj[y].pb(x);
8. }
9. **int** tim = 0;
10. vector<**int**> rnk(n + 1, 0), dfn(n + 1, 0), tree\_fa(n + 1, 0);
11. function<**void**(**int**)> dfs = [&](**int** x) {
12. rnk[x] = ++tim, dfn[tim] = x;
13. **for** (**int** y : adj[x]) {
14. **if** (!rnk[y]) {
15. tree\_fa[y] = x;
16. tree\_adj[x].pb(y);
17. dfs(y);
18. }
19. }
20. };
21. dfs(n);
22. vector<**int**> sdom(n + 1, 0), rt(n + 1, 0), home(n + 1, 0);
23. iota(all(sdom), 0);
24. iota(all(rt), 0);
25. iota(all(home), 0);
27. function<**int**(**int**)> get\_home = [&](**int** x) {
28. **if** (rt[x] == x) {
29. **return** home[x];
30. }
31. **int** tmp = get\_home(rt[x]);
32. **if** (rnk[sdom[tmp]] < rnk[sdom[home[x]]]) {
33. home[x] = tmp;
34. }
35. rt[x] = rt[rt[x]];
36. **return** home[x];
37. };
39. **for** (**int** i = n; i >= 1; i--) {
40. **int** x = dfn[i];
41. **for** (**int** y : radj[x]) {
42. **if** (rnk[y] && rnk[sdom[get\_home(y)]] < rnk[sdom[x]]) {
43. sdom[x] = sdom[get\_home(y)];
44. }
45. }
46. rt[x] = tree\_fa[x];
47. }
48. vector<**int**> dep(n + 1, 0);
49. vector<vector<**int**>> jp(20, vector<**int**>(n + 1, 0)), mn(20, vector<**int**>(n + 1, 0));
50. vector<**int**> idom(n + 1, 0);
51. vector<ll> sum(n + 1, 0);
52. function<**void**(**int**)> dfs\_tree = [&](**int** x) {
53. jp[0][x] = tree\_fa[x];
54. mn[0][x] = x;
55. **for** (**int** j = 1; j < 20; j++) {
56. jp[j][x] = jp[j - 1][jp[j - 1][x]];
57. **if** (rnk[sdom[mn[j - 1][jp[j - 1][x]]]] < rnk[sdom[mn[j - 1][x]]]) {
58. mn[j][x] = mn[j - 1][jp[j - 1][x]];
59. } **else** {
60. mn[j][x] = mn[j - 1][x];
61. }
62. }
63. **int** dt = dep[x] - dep[sdom[x]];
64. **int** ps = x, cur = x;
65. **for** (**int** j = 19; j >= 0; j--) {
66. **if** (dt >> j & 1) {
67. **if** (rnk[sdom[mn[j][cur]]] < rnk[sdom[ps]]) {
68. ps = mn[j][cur];
69. }
70. cur = jp[j][cur];
71. }
72. }
73. **if** (sdom[ps] == sdom[x]) {
74. idom[x] = sdom[x];
75. } **else** {
76. idom[x] = idom[ps];
77. }
78. sum[x] = sum[idom[x]] + x;
79. **for** (**int** y : tree\_adj[x]) {
80. dep[y] = dep[x] + 1;
81. dfs\_tree(y);
82. }
83. };
84. dfs\_tree(n);
85. **for** (**int** i = 1; i <= n; i++) {
86. cout << sum[i] << " \n"[i == n];
87. }
88. }

**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。

性质，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。

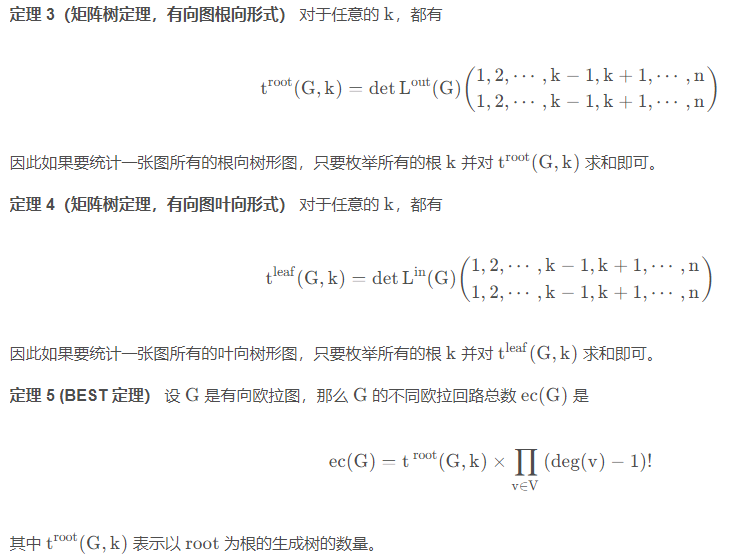
1. **int** llim = n + n - 1, nn;
2. nn = n;
3. **for**(**int** i = 1; i <= m; i++)
4. {
5. **int** x = edge[i].x;
6. **int** y = edge[i].y;
7. **int** fx = find(x);
8. **int** fy = find(y);
9. **if**(fx == fy)
10. **continue**;
11. ++n;
12. ff[fx] = n;
13. ff[fy] = n;
14. adj[n].pb(fx);
15. adj[n].pb(fy);
16. val[n] = edge[i].val;
17. **if**(n == llim)**break**;
18. }

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。

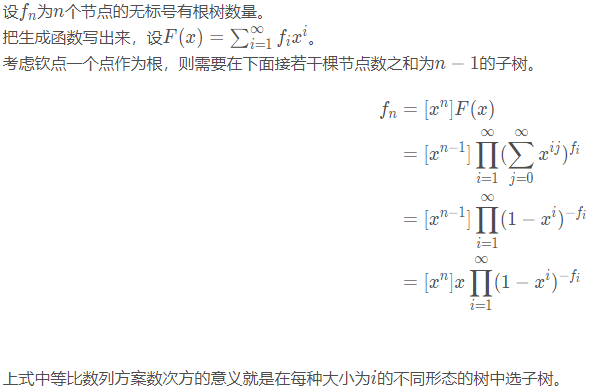
（2）.生成树计数-矩阵树定理

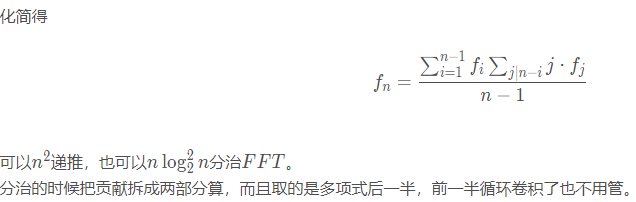
无向图：度数矩阵减去邻接矩阵，之后任意划去一行一列，求行列式的绝对值即为所求生成树个数。

有向图：

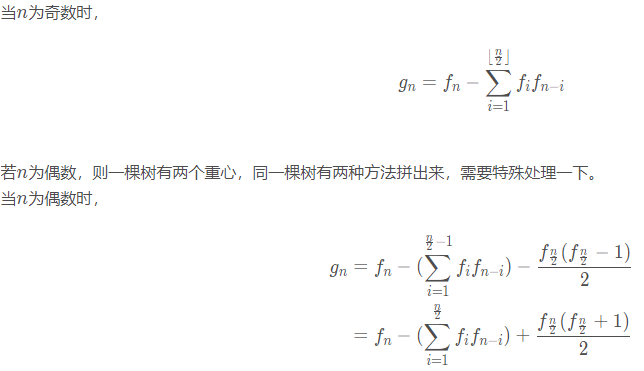


多项式做法，无标号有根树：





无标号无根树：只考虑以重心为根，进行容斥。



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)

II.可以发现每个点在prufer序列每出现一次就对应了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一。

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。

## 最小树形图

有向图的最小生成树，可能无解，需要用到左偏树

1. **const** **int** Size = 1e6 + 100;
2. **int** fa[Size], cnt, is[Size];
3. **int** find(**int** x){
4. **return** x == fa[x] ? x : fa[x] = find(fa[x]);
5. }
6. **struct** leftist{
7. **struct** point{
8. **int** l,r,d,v,t,to;
9. }a[Size]={{0, 0, -1, 0, 0, 0}};
10. **int** r[Size];
11. **void** merge(**int** &x, **int** &y) {
12. **if** (!x || !y) {
13. x = x + y;
14. **return**;
15. }
16. **if** (a[x].v > a[y].v) {
17. swap(x, y);
18. }
19. a[y].t -= a[x].t, a[y].v -= a[x].t;
20. merge(a[x].r, y);
21. **if**(a[a[x].l].d < a[a[x].r].d)
22. swap(a[x].l, a[x].r);
23. a[x].d = a[a[x].r].d + 1;
24. }
25. **void** spread(**int** &p) {
26. a[a[p].l].t += a[p].t, a[a[p].r].t += a[p].t;
27. a[a[p].l].v += a[p].t, a[a[p].r].v += a[p].t;
28. a[p].t = 0;
29. }
30. **void** pop(**int** &x) {
31. spread(x), merge(a[x].l, a[x].r), x = a[x].l;
32. }
33. point\*top(**int** &x) {
34. **while** (r[x] && find(a[r[x]].to) == x) {
35. pop(r[x]);
36. }
37. **if** (!r[x]) {
38. cout << -1 << '\n';
39. exit(0);
40. }
41. a[r[x]].to = find(a[r[x]].to);
42. **return** &a[r[x]];
43. }
44. }L;
45. **void** Union(**int** u, **int** v){
46. **if** ((u=find(u)) != (v=find(v)))
47. L.merge(L.r[v], L.r[u]), fa[u] = v;
48. }
49. **int** pre[Size];
51. **void** sol()
52. {
53. **int** n, m, r, ans = 0;
54. leftist::point \*temp;
55. cin >> n >> m >> r;
56. cnt = n, is[r] = r;
57. **for** (**int** i = 1, u, v, w; i <= m; i++) {
58. cin >> u >> v >> w;
59. L.a[i] = {0, 0, 0, w, 0, u};
60. L.merge(L.r[v], u = i);
61. }
62. **for** (**int** i = 1; i <= n << 1; i++) fa[i] = i;
63. **for** (**int** i = 1, j = i; i <= n; j = ++i) {
64. **while** (!is[j]) {
65. **while** (!is[j]) {
66. is[j] = i, j = (temp = L.top(j)) -> to;
67. ans += temp -> v;
68. }
69. **if** (is[j] != i) **break**;
70. **while** (~is[j]) {
71. is[j] = -1, j = pre[j] = (temp = L.top(j)) -> to,
72. temp -> t -= temp -> v, temp -> v = 0;
73. }
74. ++cnt;
75. **while**(is[j] != i) {
76. is[j] = i, Union(j, cnt);
77. j = pre[j];
78. }
79. j = cnt;
80. }
81. }
82. cout << ans << '\n';
83. }

# 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，比如炸飞机的某道题目，区间dp一定选最高的一个炸，然后dp到两边区间。

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，即假设i的最优决策点在k，i + 1 最优决策点在kk, kk > k，那么令g(i,j)表示i从j转移过来的大小，有g(i + 1, k) <= g(i + 1, k + 1) <= … <= g(i + 1, kk),这一类情况只要维护一个最优决策点ps，每到一个新地方尝试暴力右移即可。O(n).

（2）麻烦的情况，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，里面存的是决策点位置，以及该位置为决策点管辖的区间,也就是说，队列里一个元素记为（ps，l，r），表示根据当前已知的所有dp值，之后访问到的（l，r）区间内元素决策位置都会是ps。

显然根据决策点随i递增而单调不减的性质，这个队列实际上是单调队列，即队列中元素ps递增。

那么考虑枚举到i，要对队列如何操作：

(I).弹队首，直到队首元素的管辖区间包含i，选取队首元素的ps作为i的决策点即可算出dp（i）。

(II).把i作为决策点插入队列。因为最优决策点单调不减性，则对于队尾元素，如果对于队尾元素管辖区间的左端点，选取i都比选取队尾元素的ps优，则直接弹掉队尾元素。重复弹队尾元素，直到：

(I).队列为空，直接扔进i决策点，管辖后面所有位置即可。

(II).剩下的队尾元素，满足在队尾元素管辖的左端点还是选队尾元素更优，则二分队尾元素管辖的l，r区间，找到队尾元素管辖范围与i管辖范围的分界点，然后修改队尾元素右端点，并插入决策i。

复杂度nlogn

(3)分治优化，满足决策单调性，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方程满足可以一段一段区间合并（可能是矩阵乘积之类的），然后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. }
80. **void** sol()
81. {
82. cin >> n;
83. **for** (**int** i = 1; i < n; i++) {
84. **int** x, y;
85. cin >> x >> y;
86. adj[x].pb(y);
87. adj[y].pb(x);
88. }
89. **for** (**int** i = 1; i <= n; i++) {
90. cin >> a[i];
91. }
92. mx[0] = inf, allsz = n;
93. get\_rt(1, 0);
94. work(rt);
95. cout << res \* pw2[n] % mod << '\n';
96. }

边分治**，**类似，对于与度数相关的问题有着很大的优势，每次分治时找到一条分治中心边使这条边两端的两个联通块中较大的一个尽量小。以分治中心边为界限，恰好将当前分治的联通块中的点分成了两部分，统计路径经过分治中心边的答案，然后将分治中心边断开，递归分治中心边两端的两个联通块。

1. **inline** **void** getroot(**int** x,**int** fa)
2. {
3. size[x]=1;
4. **for**(**int** i=head[x];i;i=next[i])
5. {
6. **if**(!vis[i>>1]&&to[i]!=fa)
7. {
8. getroot(to[i],x);
9. size[x]+=size[to[i]];
10. **int** mx\_size=max(size[to[i]],sz-size[to[i]]);
11. **if**(mx\_size<num)
12. {
13. num=mx\_size;
14. root=i;
15. }
16. }
17. }
18. }

注意边分治会被菊花图卡，因此需要多叉树转二叉树，从1开始枚举每个点，对于一个点x，如果他有<=2个子节点，那么直接向子节点连边即可；否则新建两个点（++n, ++n），将x连向这两个点，并将x的子节点按奇偶分类暂时归为这两个新建点的子节点。为了不影响原树深度等信息，我们将连向新建点的边权设为0。

3.重链剖分

1. **void** dfs1(**int** pos,**int** f,**int** dp)
2. {
3. **int** mx=0;
4. deep[pos]=dp;fa[pos]=f;sz[pos]=1;
5. **for**(**int** i=0;i<mp[pos].size();i++)
6. **if**(mp[pos][i]!=f)
7. {
8. dfs1(mp[pos][i],pos,dp+1);
9. sz[pos]+=sz[mp[pos][i]];
10. **if**(sz[mp[pos][i]]>mx)mx=sz[mp[pos][i]],son[pos]=mp[pos][i];
11. }
12. }
13. **void** dfs2(**int** pos,**int** tp)
14. {
15. top[pos]=tp,dfn[++num]=pos,id[pos]=num;
16. **if**(son[pos])
17. {
18. dfs2(son[pos],tp);
19. **for**(**int** i=0;i<mp[pos].size();i++)
20. **if**(mp[pos][i]!=fa[pos]&&mp[pos][i]!=son[pos])dfs2(mp[pos][i],mp[pos][i]);
21. }
22. }

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，本质上和一般的启发式合并是一样的：将较小的块合并到较大的块，从而保证每个点至多被合并 log n 次，因为一个点每次合并后所在的块的大小至少翻倍。

然而实现的时候，我们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）

先按dfs序将所有结点排序，然后用栈维护到之前一个点为结尾的dfs链，分情况讨论，因为新来的点dfs序一定大于栈顶的，所以要么它和栈顶的lca是栈顶本身，要么则分立两边，那么如果是第一种就继续接下去，第二种则一直将栈弹至栈顶深度小于等于lca，再接上。

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. {
5. **int** tt=k,nw,f;
6. sort(imp+1,imp+k+1,cmp);hd=0;
7. **for**(**int** i=1;i<=k;i++)
8. {
9. nw=imp[i];
10. **if**(!hd){fa[nw]=0,S[++hd]=nw;**continue**;}
11. f=lca(nw,S[hd]);
12. **while**(dep[S[hd]]>dep[f])
13. {
14. **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
15. --hd;
16. }
17. **if**(f!=S[hd])  //一个新的关键点之间的lca加入
18. {
19. imp[++tt]=f;
20. fa[f]=S[hd];
21. S[++hd]=f;
22. }
23. fa[nw]=f,S[++hd]=nw;
24. }
25. k=tt;sort(imp+1,imp+k+1,cmp);  //新加了关键点，重新sort
26. }

5树上线段树合并：（也可见Sam处代码）

一般每个节点先开一个点的动态开点线段树，让后dfs整棵树的同时将线段树逐层合并上传，在每个点处理询问。

1. **int** new\_node()
2. {++snum,seg[snum].ls=seg[snum].rs=seg[snum].w=0;**return** snum;}
3. **void** push\_up(**int** x)
4. {seg[x].w=seg[seg[x].ls].w+seg[seg[x].rs].w;}
5. **int** build(**int** l,**int** r,**int** to)
6. {
7. **int** nw=new\_node(),mid;
8. **if**(l==r)seg[nw].w++;
9. **else**
10. {
11. mid=(l+r)>>1;
12. **if**(to<=mid)seg[nw].ls=build(l,mid,to);
13. **else** seg[nw].rs=build(mid+1,r,to);
14. push\_up(nw);
15. }
16. **return** nw;
17. }
18. **void** ins(**int** nw,**int** l,**int** r,**int** to)
19. {
20. **int** mid;
21. **if**(l==r)seg[nw].w++;
22. **else**
23. {
24. mid=(l+r)>>1;
25. **if**(to<=mid)
26. {
27. **if**(!seg[nw].ls)seg[nw].ls=new\_node();
28. ins(seg[nw].ls,l,mid,to);
29. }
30. **else**
31. {
32. **if**(!seg[nw].rs)seg[nw].rs=new\_node();
33. ins(seg[nw].rs,mid+1,r,to);
34. }
35. push\_up(nw);
36. }
37. }
38. **int** mg(**int** x,**int** y)
39. {
40. **if**(!x||!y)**return** x+y;
41. seg[x].w=seg[x].w+seg[y].w;
42. **if**(seg[x].ls||seg[y].ls)seg[x].ls=mg(seg[x].ls,seg[y].ls);
43. **if**(seg[x].rs||seg[y].rs)seg[x].rs=mg(seg[x].rs,seg[y].rs);
44. **return** x;
45. }
46. **void** Del(**int** nw,**int** l,**int** r,**int** to)
47. {
48. **int** mid;
49. **if**(l==r)seg[nw].w--;
50. **else**
51. {
52. mid=(l+r)>>1;
53. **if**(to<=mid) Del(seg[nw].ls,l,mid,to);
54. **else** Del(seg[nw].rs,mid+1,r,to);
55. push\_up(nw);
56. }
57. }
58. **int** ask(**int** nw,**int** l,**int** r,**int** to)
59. {
60. **if**(l==r&&to==l)**return** seg[nw].w;
61. **else**
62. {
63. **int** mid=(l+r)>>1;
64. **if**(to<=mid&&seg[nw].ls)**return** ask(seg[nw].ls,l,mid,to);
65. **if**(to>mid&&seg[nw].rs)**return** ask(seg[nw].rs,mid+1,r,to);
66. **return** 0;
67. }
68. }
69. **void** dfs2(**int** pos,**int** fa)
70. {
71. **for**(**int** i=0;i<add[pos].size();i++)
72. {
73. **if**(!rt[pos])rt[pos]=build(1,2\*n,add[pos][i]+n);
74. **else** ins(rt[pos],1,2\*n,add[pos][i]+n);
75. }
76. **for**(**int** i=0;i<mp[pos].size();i++)
77. {
78. **if**(mp[pos][i]!=fa)
79. {
80. dfs2(mp[pos][i],pos);
81. **if**(!rt[pos])rt[pos]=rt[mp[pos][i]];
82. **else** rt[pos]=mg(rt[pos],rt[mp[pos][i]]);
83. }
84. }
85. **for**(**int** i=0;i<del[pos].size();i++)
86. Del(rt[pos],1,2\*n,del[pos][i]+n);
87. ans[pos]+=ask(rt[pos],1,2\*n,dep[pos]+a[pos]+n);
88. **if**(a[pos]!=0)ans[pos]+=ask(rt[pos],1,2\*n,dep[pos]-a[pos]+n);
89. }

# 十四、分治、分块、莫队

1. 维护序列分块：
2. **void** build(**int** n)//建立各块(各个分块这个操作大致一样)
3. {
4. block=sqrt(n);//大小
5. num=n/block;**if**(n%block)num++;//如果不整除显然数量加个1
6. **for**(**int** i=1;i<=num;i++)
7. L[i]=(i-1)\*block+1,R[i]=i\*block;//左右端点
8. R[num]=n;   //最后一个右端点一定是结尾
9. **for**(**int** i=1;i<=n;i++)
10. belong[i]=(i-1)/block+1;    //belong数组
11. }

2.对询问分块：

主要是一种思路，每根号n次修改操作统一执行修改，然后对于查询，先得到最近一次统一修改的答案，然后再考察不超过根号n次修改带来的影响。

3.整除分块：（下为杜教筛里的）

1. **for**(**int** l=1,r;l<=H;l=r+1)
2. {
3. r=H/(H/l);
4. **if**(L>=l)r=min(r,L/(L/l));
5. ans=(1LL\*qpow(H/l-L/l,n)\*(ask(r)-ask(l-1))%mod+ans)%mod;
6. }

4.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。

5.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），就是对于每一个插入操作处理出它存在的时间，那么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。而一般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往下传......

6.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。复杂度n\*log^2，优点在于较好写并且常数小。

# 十五、概率期望、高斯消元

期望这个东西个人理解就是sigma概率\*贡献，它满足线性性：对于两个不相关的随机变量x,y,E(x±y)=E(x)0±E(y),E(xy)=E(x)E(y),E(x/y)=E(x)/E(y).

期望dp，一般是n元一次线性方程组，套高斯消元的板子，有的需要线代知识推一下。

高斯消元模板：

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**

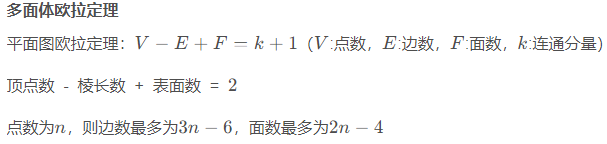
**bst.flip() 逐位取反**

**bst.count() 返回1的个数**

# 十七、计算几何常用函数

1. #include<bits/stdc++.h>
2. #define pii pair<int,int>
3. #define fi first
4. #define sc second
5. #define pb push\_back
6. #define ll long long
7. #define trav(v,x) for(auto v:x)
8. #define all(x) (x).begin(), (x).end()
9. #define VI vector<int>
10. #define VLL vector<ll>
11. #define pll pair<ll, ll>
12. #define double long double
13. //#define int long long
14. **using** **namespace** std;
16. #ifdef LOCAL
17. **void** debug\_out(){cerr << endl;}
18. **template**<**typename** Head, **typename**... Tail>
19. **void** debug\_out(Head H, Tail... T)
20. {
21. cerr << " " << to\_string(H);
22. debug\_out(T...);
23. }
24. #define debug(...) cerr << "[" << #\_\_VA\_ARGS\_\_ << "]:", debug\_out(\_\_VA\_ARGS\_\_)
25. #else
26. #define debug(...) 42
27. #endif
29. **const** **int** N = 1e6 + 100;
30. **const** ll inf = 1e18;
31. **const** ll mod = 998244353;//1e9 + 7;
33. **using** ld = ll;
34. **const** **double** eps = 1e-14;
35. **const** **double** pi = acosl(-1);
36. **int** sgn(ld x) {
37. **return** x < -eps ? -1 : x > eps;
38. }
39. **int** cmp(ld x, ld y) {
40. **return** sgn(x - y);
41. }
43. **using** T = ld;
44. **struct** Point { // 点
45. T x, y;
46. Point(T x = 0, T y = 0) : x(x), y(y) {}
47. **bool** operator<(Point B) **const**{ // x第一关键字
48. **return** x == B.x ? y < B.y : x < B.x;
49. }
50. **bool** operator==(Point B) **const**{
51. **return** !sgn(x - B.x) && !sgn(y - B.y);
52. }
53. **bool** operator <=(Point B) **const** {
54. **return** ((\***this**) < B || (\***this**) == B);
55. }
56. Point operator+(Point B) **const**{
57. **return** Point(x + B.x, y + B.y);
58. }
59. Point operator-(Point B) **const**{
60. **return** Point(x - B.x, y - B.y);
61. }
62. Point operator\*(T a) **const**{ // 标量乘
63. **return** Point(x \* a, y \* a);
64. }
65. Point operator/(T a) **const**{ // 标量除
66. **return** Point(x / a, y / a);
67. }
68. T operator\*(Point B) **const**{ // 点积
69. **return** x \* B.x + y \* B.y;
70. }
71. T operator^(Point B) **const**{ // 叉积模长
72. **return** x \* B.y - y \* B.x;
73. }
74. Point operator-() **const**{    // 取负,关于原点对称
75. **return** Point(-x, -y);
76. }
77. ld angle() {    // 反正切,与x轴方位角, (-pi, pi]
78. **return** atan2l(**this**->y, **this**->x);
79. }
80. T length2() { // 视为原点到(x,y)向量，模长的平方
81. **return** x \* x + y \* y;
82. }
83. ld length() { // 模长
84. **return** sqrtl(length2());
85. }
86. Point unit() {  // 单位方向向量
87. **return** \***this** / **this** -> length();
88. }
89. Point normal() {  // 单位法向量
90. **return** Point(-y, x) / **this** -> length();
91. }
92. Point trunc(ld r) {  // 化为长度为r的向量
93. ld l = length();
94. **if** (!sgn(l))
95. **return** \***this**;
96. r /= l;
97. **return** Point(x \* r, y \* r);
98. }
99. Point conj() { // 共轭向量
100. **return** Point(x, -y);
101. }
102. Point to\_left() { // 绕原点左转90度
103. **return** Point(-y, x);
104. }
105. Point to\_right() { // 绕原点右转90度
106. **return** Point(y, -x);
107. }
108. Point rotate(ld rad) {  // 逆时针旋转 rad 弧度
109. **return** Point(x \* cos(rad) - y \* sin(rad), x \* sin(rad) + y \* cos(rad));
110. }
111. **friend** **int** relation(Point a, Point b, Point c) { // c是否在(a,b)的逆时针侧
112. **return** sgn((b - a) ^ (c - a));
113. }
114. **friend** ld get\_angle(Point a, Point b) { // 向量夹角
115. **return** acosl((a \* b) / a.length() / b.length());
116. }
117. **friend** T area(Point a, Point b, Point c) { //
118. **return** fabsl((b - a) ^ (c - a)); // (a,b)(a,c)平行四边形面积
119. }
120. **friend** T get\_dis2(Point a, Point b) { // 两间距离方
121. **return** (a - b).length2();
122. }
123. **friend** ld get\_dis(Point a, Point b) { // 两点距离
124. **return** sqrtl(get\_dis2(a, b));
125. }
126. **friend** ld project(Point a, Point b, Point c) {  // 求向量ac在向量ab上的投影长度
127. **return** ((b - a) \* (c - a)) / (b - a).length();
128. }
129. **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)
130. **return** y > 0 || (y == 0 && x >= 0);
131. }
132. **friend** ostream& operator<<(ostream& os, Point a) {
133. **return** os << "(" << a.x << ',' << a.y << ')';
134. }
135. };
137. **bool** argcmp(Point a, Point b) { // 关于原点极角排序,一二三四象限顺序
138. **if** (a.up() != b.up())
139. **return** a.up() > b.up();
140. **return** (a ^ b) == 0 ? a.x < b.x : (a ^ b) > 0;
141. }
142. Point rotate(Point a, Point bas, ld theta) { // a点绕bas点逆时针转theta弧度
143. **return** (a - bas).rotate(theta) + bas;
144. }
145. Point reflect(Point v, Point l) { // 光线v照射到平面l后反射
146. Point res;
147. Point E = l / l.length();  // 单位向量
148. T d = E \* v;
149. **return** (E \* 2 \* d - v);
150. }
152. **struct** Line { // 直线
153. Point p, v; // p为直线上一点,v为方向向量
154. ld rad; // rad为方向角 (-pi, pi]
155. //亦可表示向量v逆时针方向的半平面
156. Line(){}
157. Line(Point p, Point v):p(p), v(v) {
158. rad = atan2l(v.y, v.x);
159. }
160. Point get\_point(ld t) {
161. **return** p + v \* t;
162. }
163. **int** under(Point a) { // 射线是否在点a下方
164. **return** relation(p, p + v, a);
165. }
166. **bool** operator <(Line b) { // 比较
167. **if** (!cmp(rad, b.rad)) {
168. **return** under(b.p) < 0; // 靠上侧的排前面
169. }
170. **return** rad < b.rad;
171. }
172. ld dis2point(Point a) {  // 点a到直线的距离
173. **return** fabsl((v ^ (a - p)) / v.length());
174. }
175. Point foot(Point a) { // 点在直线的投影点(垂足)
176. **return** p + v \* (v \* (a - p) / v.length2());
177. }
178. Point symmetry(Point a) { // 点关于直线对称的点(即镜面反射)
179. **return** foot(a) \* 2 - a;
180. }
181. **bool** parallel(Line b) { // 两直线是否平行(共线也算)
182. **if** (!sgn(v ^ b.v)) {
183. **return** 1;
184. }
185. **return** 0;
186. }
187. **bool** colinear(Line b) { // 两直线是否共线
188. **if** (parallel(b)) {
189. **if** (!sgn((b.p - p) ^ v)) {
190. **return** 1;
191. }
192. }
193. **return** 0;
194. }
195. Point intersect(Line b) {  // 两直线交点(不能平行)
196. assert(!parallel(b));
197. Point u = p - b.p;
198. T t = (b.v ^ u) / (v ^ b.v);
199. **return** get\_point(t);
200. }
201. };
203. **struct** Segment { // 线段
204. Point a, b;
205. Segment(){}
206. Segment(Point aa, Point bb) {
207. **if** (bb < aa) {
208. swap(aa, bb);
209. }
210. a = aa, b = bb;
211. }
212. **bool** isIntersect(Point p) { // 点p是否在线段上
213. **return** !sgn((a - p) ^ (b - p)) && sgn((a - p) \* (b - p)) <= 0;
214. }
215. **bool** parallel(Segment seg) {
216. **return** Line(a, b - a).parallel(Line(seg.a, seg.b - seg.a));
217. }
218. **bool** colinear(Segment seg)  { // 两线段是否共线, 有重合部分
219. // 只考虑seg退化的情况
220. **if** (seg.a == seg.b) {
221. **if** (Line(a, b - a).under(seg.a) == 0) {
222. **if** (a <= seg.a && seg.a <= b) {
223. **return** 1;
224. }
225. }
226. **return** 0;
227. }
228. **if** (Line(a, b - a).colinear(Line(seg.a, seg.b - seg.a))) {
229. **if** ((a <= seg.a && seg.a <= b) || (seg.a <= a && a <= seg.b)) {
230. **return** 1;
231. }
232. }
233. **return** 0;
234. }
235. **bool** isIntersect(Segment seg) { // 两线段是否相交(含端点含共线)
236. **if** (parallel(seg)) { //线段平行
237. **return** colinear(seg);
238. }
239. **const** Point &a1 = a, &a2 = b, &b1 = seg.a, &b2 = seg.b;
240. T c1 = (a2 - a1) ^ (b1 - a1), c2 = (a2 - a1) ^ (b2 - a1);
241. T c3 = (b2 - b1) ^ (a1 - b1), c4 = (b2 - b1) ^ (a2 - b1);
242. **return** sgn(c1) \* sgn(c2) <= 0 && sgn(c3) \* sgn(c4) <= 0;
243. }
244. **bool** isIntersect(Line l) { // 线段和直线是否相交 (含端点)
245. **return** l.under(a) \* l.under(b) <= 0;
246. }
247. ld dis2point(Point p) { // 点到线段距离
248. **if** (sgn((p - a) \* (b - a)) < 0 || sgn((p - b) \* (a - b)) < 0) {
249. **return** min(get\_dis(b, p), get\_dis(a, p));
250. }
251. **return** Line(a, b - a).dis2point(p);
252. }
253. ld dis2seg(Segment seg) {
254. **if** (isIntersect(seg)) {
255. **return** 0;
256. }
257. **return** min({dis2point(seg.a), dis2point(seg.b), seg.dis2point(a), seg.dis2point(b)});
258. }
259. };
261. ld get\_area(vector<Point> pts) {
262. ld res = 0;
263. **for** (**int** i = 1; i + 1 < pts.size(); i++) {
264. res += (pts[i] - pts[0]) ^ (pts[i + 1] - pts[0]);
265. }
266. **return** res / 2;
267. }
269. vector<Point> half\_plane\_intersect(vector<Line> lines) {
270. sort(all(lines));
271. deque<Point> pts;
272. deque<Line> ls;
273. **for** (auto l : lines) {
274. **if** (ls.empty()) {
275. ls.pb(l);
276. **continue**;
277. }
278. **while**(!pts.empty() && l.under(pts.back()) <= 0) {
279. pts.pop\_back();
280. ls.pop\_back();
281. }
282. **while**(!pts.empty() && l.under(pts[0]) <= 0) {
283. pts.pop\_front();
284. ls.pop\_front();
285. }
286. **if** (!sgn(ls.back().v ^ l.v)) {
287. **if** (sgn(ls.back().v \* l.v) > 0) {
288. **continue**;
289. } **else** {
290. **return** vector<Point>{};
291. }
292. }
293. pts.pb(l.intersect(ls.back()));
294. ls.pb(l);
295. }
296. **while**(!pts.empty() && ls[0].under(pts.back()) <= 0) {
297. pts.pop\_back();
298. ls.pop\_back();
299. }
300. **if** (ls.size() > 2) {
301. pts.push\_back(ls[0].intersect(ls.back()));
302. }
303. **return** vector<Point>(pts.begin(), pts.end());
304. }
306. vector<Line> get\_half\_plane(Point u, Segment v) {
307. **if** (((v.a - u) ^ (v.b - u)) == 0) {
308. **return** vector<Line>{Line(u, v.a - u), Line(v.a, u - v.a)};
309. }
310. **if** (((v.a - u) ^ (v.b - u)) < 0) {
311. swap(v.a, v.b);
312. }
313. **return** vector<Line> {
314. Line(v.a, u - v.a),
315. Line(v.a, v.b - v.a),
316. Line(u, v.b - u)
317. };
318. }
320. // (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
321. **int** point\_in\_polygon(**const** Point& p, vector<Point>& poly) {
322. **int** wn = 0;
323. **int** n = poly.size();
324. **for** (**int** i = 0; i < n; i++) {
325. **if** (Segment(poly[i], poly[(i + 1) % n]).isIntersect(p))
326. **return** -1;
327. **int** k = sgn((poly[(i + 1) % n] - poly[i]) ^ (p - poly[i]));
328. **int** d1 = sgn(poly[i].y - p.y);
329. **int** d2 = sgn(poly[(i + 1) % n].y - p.y);
330. **if** (k > 0 && d1 <= 0 && d2 > 0)
331. wn++;
332. **if** (k < 0 && d2 <= 0 && d1 > 0)
333. wn--;
334. }
335. **if** (wn != 0)
336. **return** 1;
337. **return** 0;
338. }
340. **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
341. **if** (a.empty())
342. **return** **false**;
343. **int** l = 1, r = a.size() - 1;
344. **while** (l <= r) {
345. **int** mid = l + r >> 1;
346. **double** ls = (a[mid] - a[0]) ^ (p - a[0]);
347. **double** rs = (a[mid + 1] - a[0]) ^ (p - a[0]);
348. **if** (ls >= 0 && rs <= 0) {
349. **int** type = sgn((a[mid + 1] - a[mid]) ^ (p - a[mid]));
350. **if** (type == 0)
351. **return** -1;
352. **else** **if** (type == 1)
353. **return** 1;
354. **return** 0;
355. } **else** **if** (ls < 0) {
356. r = mid - 1;
357. } **else** {
358. l = mid + 1;
359. }
360. }
361. **return** **false**;
362. }
364. **struct** Circle {
365. Point p;
366. **double** r;
367. Circle(Point \_p = Point(0, 0), **double** \_r = 0) : p(\_p), r(\_r) {}
368. // 三角形外接圆
369. Circle(Point a, Point b, Point c) {
370. Line u = Line({(a + b) / 2}, {(b - a).rotate(pi / 2)});
371. Line v = Line({(a + c) / 2}, {(c - a).rotate(pi / 2)});
372. p = u.intersect(v);
373. r = get\_dis(p, a);
374. }
375. // 三角形内切圆(bool t 只是为了与外接圆区别)
376. Circle(Point a, Point b, Point c, **bool** t) {
377. Line u, v;
378. **double** m = atan2l(b.y - a.y, b.x - a.x), n = atan2l(c.y - a.y, c.x - a.x);
379. u.p = a;
380. u.v = u.p + Point(cos((n + m) / 2), sin((n + m) / 2));
381. v.p = b;
382. m = atan2l(a.y - b.y, a.x - b.x), n = atan2l(c.y - b.y, c.x - b.y);
383. v.v = v.p + Point(cos((n + m) / 2), sin((n + m) / 2));
384. p = u.intersect(v);
385. r = Line(a, b).dis2point(p);
386. }
387. **bool** operator==(Circle v) {
388. **return** (p == v.p) && sgn(r - v.r) == 0;
389. }
390. **bool** operator<(Circle v) **const** {
391. **return** ((p < v.p) || ((p == v.p) && sgn(r - v.r) < 0));
392. }
393. **double** area() {
394. **return** pi \* r \* r;
395. }
396. **double** length() {
397. **return** 2 \* pi \* r;
398. }
399. // 点和圆的关系        -1圆内   0圆上   1圆外
400. **int** relation(Point a) {
401. **double** dist = get\_dis(p, a);
402. **if** (sgn(dist - r) < 0)
403. **return** -1;
404. **else** **if** (sgn(dist - r) == 0)
405. **return** 0;
406. **return** 1;
407. }
408. // 直线和圆的关系     -1相交   0相切    1相离
409. **int** line\_relation(Line v) {
410. **double** dist = v.dis2point(p);
411. **if** (sgn(dist - r) < 0)
412. **return** -1;
413. **else** **if** (sgn(dist - r) == 0)
414. **return** 0;
415. **else**
416. **return** 1;
417. }
418. // 两圆的关系  5 相离   4 外切   3 相交   2内切    1 内含
419. **int** circle\_relation(Circle v) {
420. **double** dist = get\_dis(p, v.p);
421. **if** (sgn(dist - r - v.r) > 0)
422. **return** 5;
423. **if** (sgn(dist - r - v.r) == 0)
424. **return** 4;
425. **double** l = fabs(r - v.r);
426. **if** (sgn(dist - r - v.r) < 0 && sgn(dist - l) > 0)
427. **return** 3;
428. **if** (sgn(dist - l) == 0)
429. **return** 2;
430. **if** (sgn(dist - l) < 0)
431. **return** 1;
432. **return** -1;
433. }
434. // 求两个圆的交点，并返回交点个数
435. **int** cross\_circle(Circle v, Point& p1, Point& p2) {
436. **int** rel = circle\_relation(v);
437. **if** (rel == 1 || rel == 5)
438. **return** 0;
439. **double** d = get\_dis(p, v.p);
440. **double** l = (d \* d + r \* r - v.r \* v.r) / (d \* 2);
441. **double** h = sqrtl(r \* r - l \* l);
442. Point tmp = p + (v.p - p).trunc(l);
443. p1 = tmp + ((v.p - p).to\_left().trunc(h));
444. p2 = tmp + ((v.p - p).to\_right().trunc(h));
445. **if** (rel == 2 || rel == 4)
446. **return** 1;
447. **return** 2;
448. }
449. // 求直线和圆的交点，返回交点个数
450. **int** cross\_line(Line v, Point& p1, Point& p2) {
451. **if** ((\***this**).line\_relation(v) == 1)
452. **return** 0;
453. Point a = v.foot(p);
454. **double** d = v.dis2point(p);
455. d = sqrtl(r \* r - d \* d);
456. **if** (sgn(d) == 0) {
457. p1 = a, p2 = a;
458. **return** 1;
459. }
460. p1 = a + v.v.trunc(d);
461. p2 = a - v.v.trunc(d);
462. **return** 2;
463. }
464. // 过一点作圆的切线(先判断点和圆的关系)
465. **int** tangent\_line(Point q, Line& u, Line& v) {
466. **int** x = relation(q);
467. **if** (x == -1)
468. **return** 0;
469. **if** (x == 0) {
470. u = Line(q, (p - q).to\_left());
471. v = u;
472. **return** 1;
473. }
474. **double** d = get\_dis(p, q);
475. **double** rad = asin(r / d);
476. u = Line(q, (p - q).rotate(rad));
477. v = Line(q, (p - q).rotate(-rad));
478. **return** 2;
479. }
480. // 求两圆相交面积
481. **double** circle\_cross\_area(Circle v) {
482. **int** rel = circle\_relation(v);
483. **if** (rel >= 4)
484. **return** 0;
485. **if** (rel <= 2)
486. **return** min(area(), v.area());
487. **double** d = get\_dis(p, v.p);
488. **double** hf = (r + v.r + d) / 2;
489. **double** ss = 2 \* sqrtl(hf \* (hf - r) \* (hf - v.r) \* (hf - d));
490. **double** a1 = acos((r \* r + d \* d - v.r \* v.r) / (2.0 \* r \* d));
491. a1 = a1 \* r \* r;
492. **double** a2 = acos((v.r \* v.r + d \* d - r \* r) / (2.0 \* v.r \* d));
493. a2 = a2 \* v.r \* v.r;
494. **return** a1 + a2 - ss;
495. }
496. // 得到过a,b两点, 半径为r1的两个圆
497. **friend** **int** get\_circle(Point a, Point b, **double** r1, Circle& c1, Circle& c2) {
498. Circle x(a, r1), y(b, r1);
499. **int** t = x.cross\_circle(y, c1.p, c2.p);
500. **if** (!t)
501. **return** 0;
502. c1.r = c2.r = r1;
503. **return** t;
504. }
505. };
507. **template** <**class** T>
508. **struct** convex {
509. vector<Point> q;
510. convex() {}
511. convex(vector<Point>& B) : q(B) {}
512. convex(**const** convex& B) : q(B.q) {}
513. convex& operator=(**const** convex& B) {
514. q = B.q;
515. **return** \***this**;
516. }
517. Point& operator[](**int** x) noexcept {
518. **return** q[x];
519. }
520. **int** size() **const** {
521. **return** q.size();
522. }
523. **int** nxt(**int** x) **const** {
524. **return** x == size() - 1 ? 0 : x + 1;
525. }
526. **int** pre(**int** x) **const** {
527. **return** x == 0 ? size() - 1 : x - 1;
528. }
529. **void** init(vector<Point>& v) {
530. sort(v.begin(), v.end());
531. **int** n = v.size(), top = 0;
532. vector<**int**> st(n + 10);
533. **for** (**int** i = 0; i < n; i++) {
534. **while** (top > 1 && sgn((v[st[top]] - v[st[top - 1]]) ^ (v[i] - v[st[top - 1]])) <= 0)
535. top--;
536. st[++top] = i;
537. }
538. **int** k = top;
539. **for** (**int** i = n - 2; i >= 0; i--) {
540. **while** (top > k && sgn((v[st[top]] - v[st[top - 1]]) ^ (v[i] - v[st[top - 1]])) <= 0)
541. top--;
542. st[++top] = i;
543. }
544. **for** (**int** i = 1; i < top; i++)
545. q.push\_back(v[st[i]]);
546. **return**;
547. }
548. **double** get\_length() {
549. **double** res = 0;
550. **for** (**int** i = 0; i < size(); i++)
551. res += get\_dist(q[i], q[nxt(i)]);
552. **return** res;
553. }
554. T get\_area2() {
555. T res = 0;
556. **for** (**int** i = 0; i < size(); i++)
557. res += (q[i] ^ q[nxt(i)]);
558. **return** abs(res);
559. }
560. Point getBaryCentre() **const** {  // 重心
561. Point res(0, 0);
562. **double** are = 0;
563. **const** **int** sz = size();
564. **for** (**int** i = 1; i < sz - 1; i++) {
565. **double** tmp = (q[i] - q[0]) ^ (q[i + 1] - q[0]);
566. **if** (!sgn(tmp))
567. **continue**;
568. are += tmp;
569. res.x += (q[0].x + q[i].x + q[i + 1].x) / 3 \* tmp;
570. res.y += (q[0].y + q[i].y + q[i + 1].y) / 3 \* tmp;
571. }
572. **if** (sgn(are))
573. res = res / are;
574. **return** res;
575. }
576. vector<T> sum;
577. **void** get\_sum() {
578. vector<T> a(q.size());
579. **for** (**int** i = 0; i < q.size(); i++)
580. a[i] = q[pre(i)] ^ q[i];
581. sum.resize(q.size());
582. partial\_sum(a.begin(), a.end(), sum.begin());
583. }
584. T query\_sum(**int** l, **int** r) **const** {
585. **if** (l <= r)
586. **return** sum[r] - sum[l] + (q[r] ^ q[l]);
587. **return** sum[size() - 1] - sum[l] + sum[r] + (q[r] ^ q[l]);
588. }
590. // 闵可夫斯基和
591. convex operator+(**const** convex& B) **const** {
592. **const** auto& a = **this**->q;
593. **const** auto& b = B.q;
594. **int** n = q.size(), m = b.size();
595. Point sa = q[0], sb = b[0];
596. **for** (**int** i = 0; i < n; i++) {
597. **if** (a[i].y < sa.y || (a[i].y == sa.y && a[i].x < sa.x))
598. sa = a[i];
599. }
600. **for** (**int** i = 0; i < m; i++) {
601. **if** (b[i].y < sb.y || (b[i].y == sb.y && b[i].x < sb.x)) {
602. sb = b[i];
603. }
604. }
605. auto s = sa + sb;
606. vector<Point> d(n + m);
607. **for** (**int** i = 0; i < n; i++)
608. d[i] = a[(i + 1) % n] - a[i];
609. **for** (**int** i = 0; i < m; i++)
610. d[n + i] = b[(i + 1) % m] - b[i];
611. sort(d.begin(), d.end(), [&](**const** Point& A, **const** Point& B) {
612. **if** (A.up() ^ B.up())
613. **return** A.up() > B.up();
614. **return** (A ^ B) > 0;
615. });
616. vector<Point> c(n + m);
617. c[0] = s;
618. **for** (**int** i = 0; i < n + m - 1; i++)
619. c[i + 1] = c[i] + d[i];
620. convex res;
621. res.init(c);
622. **return** res;
623. }
625. // 旋转卡壳
626. **template** <**class** F>
627. **void** rotating\_calipres(**const** F& func) **const** {
628. **for** (**int** i = 0, j = 1; i < q.size(); i++) {
629. auto d = q[i], e = q[nxt(i)];
630. func(d, e, q[j]);
631. **while** (area(d, e, q[j]) <= area(d, e, q[nxt(j)])) {
632. j = nxt(j);
633. func(d, e, q[j]);
634. }
635. }
636. }
638. // 凸包直径(平方)
639. T diameter2() **const** {
640. **if** (q.size() == 1)
641. **return** 0;
642. **if** (q.size() == 2)
643. **return** get\_dis2(q[0], q[1]);
644. T ans = 0;
645. auto func = [&](**const** Point& a, **const** Point& b, **const** Point& c) {
646. ans = max({ans, get\_dis2(a, c), get\_dis2(b, c)});
647. };
648. rotating\_calipres(func);
649. **return** ans;
650. }
652. // 凸多边形关于某一方向的极点, 复杂度 O(logn)
653. **template** <**class** F>
654. **int** extreme(**const** F& dir) **const** {
655. **const** auto check = [&](**const** **int** i) {
656. **return** sgn(dir(q[i]) ^ (q[nxt(i)] - q[i])) >= 0;
657. };
658. **const** auto dir0 = dir(q[0]);
659. **const** auto check0 = check(0);
660. **if** (!check0 && check(**this**->size() - 1))
661. **return** 0;
662. **const** auto cmp = [&](**const** Point& v) {
663. **const** **int** vi = &v - q.data();
664. **if** (vi == 0)
665. **return** 1;
666. **const** auto checkv = check(vi);
667. **const** auto t = sgn(dir0 ^ (v - q[0]));
668. **if** (vi == 1 && checkv == check0 && sgn(dir0 ^ (v - q[0])) == 0)
669. **return** 1;
670. **return** checkv ^ (checkv == check0 && t <= 0);
671. };
672. **return** partition\_point(q.begin(), q.end(), cmp) - q.begin();
673. }
675. // 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)
676. // 必须保证点在多边形外
677. pair<**int**, **int**> tangent(**const** Point& a) **const** {
678. **const** **int** i = extreme([&](**const** Point& u) {
679. **return** u - a;
680. });
681. **const** **int** j = extreme([&](**const** Point& u) {
682. **return** a - u;
683. });
684. **return** {i, j};
685. }
687. // 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)
688. pair<**int**, **int**> tangent(**const** Line& a) **const** {
689. **const** **int** i = extreme([&](...) {
690. **return** a.v;
691. });
692. **const** **int** j = extreme([&](...) {
693. **return** -a.v;
694. });
695. **return** {i, j};
696. }
698. **friend** **int** inConvex(**const** Point& p, **const** convex& c) {
699. **return** inConvex(p, c.q);
700. }
701. };
702. **using** Convex = convex<ld>;
704. **using** \_T = ld;
705. pair<\_T, \_T> minmax\_triangle(**const** vector<Point>& vec) { //最小最大三角形面积
706. **if** (vec.size() <= 2)
707. **return** {0, 0};
708. **const** \_T tmpans = abs((vec[0] - vec[1]) ^ (vec[0] - vec[2]));
709. \_T maxans = tmpans, minans = tmpans;
711. vector<pair<**int**, **int**>> evt;
712. evt.reserve(vec.size() \* vec.size());
714. **for** (**signed** i = 0; i < vec.size(); i++) {
715. **for** (**signed** j = 0; j < vec.size(); j++) {
716. **if** (i == j || vec[i] == vec[j])
717. **continue**;
718. evt.push\_back({i, j});
719. }
720. }
721. sort(evt.begin(), evt.end(), [&](**const** pair<**int**, **int**>& u, **const** pair<**int**, **int**>& v) {
722. **const** Point du = vec[u.second] - vec[u.first], dv = vec[v.second] - vec[v.first];
723. **return** argcmp({du.y, -du.x}, {dv.y, -dv.x});
724. });
725. vector<**signed**> vx(vec.size()), pos(vec.size());
726. **for** (**signed** i = 0; i < vec.size(); i++)
727. vx[i] = i;
728. sort(vx.begin(), vx.end(), [&](**int** x, **int** y) {
729. **return** vec[x] < vec[y];
730. });
731. **for** (**signed** i = 0; i < vx.size(); i++)
732. pos[vx[i]] = i;
733. **for** (auto [u, v] : evt) {
734. **signed** i = pos[u], j = pos[v];
735. **if** (i > j)
736. swap(u, v), swap(i, j);
737. **const** Point vecu = vec[u], vecv = vec[v];
738. **if** (i > 0)
739. minans = min(minans, abs((vec[vx[i - 1]] - vecu) ^ (vec[vx[i - 1]] - vecv)));
740. **if** (j < vx.size() - 1)
741. minans = min(minans, abs((vec[vx[j + 1]] - vecu) ^ (vec[vx[j + 1]] - vecv)));
742. maxans = max({maxans, abs((vec[vx[0]] - vecu) ^ (vec[vx[0]] - vecv)), abs((vec[vx.back()] - vecu) ^ (vec[vx.back()] - vecv))});
743. swap(vx[i], vx[j]);
744. pos[u] = j, pos[v] = i;
745. }
746. **return** {minans, maxans};
747. }
749. /\* 动态凸包 1.插入一个点   2.查询某个点是否被凸包包含(包括边界) \*/
750. **struct** Hull {
751. set<Point> su, sd;
752. **bool** query(set<Point>& s, Point u, **int** flag) {
753. auto l = s.upper\_bound(u), r = s.lower\_bound(u);
754. **if** (r == s.end() || l == s.begin())
755. **return** **false**;
756. l--;
757. **return** relation(\*l, u, \*r) \* flag >= 0;
758. }
759. **void** insert(set<Point>& s, Point u, **int** flag) {
760. **if** (query(s, u, flag))
761. **return**;
762. auto it = s.insert(u).first;
763. **while** (**true**) {
764. auto mid = it;
765. **if** (mid == s.begin())
766. **break**;
767. auto l = --mid;
768. **if** (l == s.begin())
769. **break**;
770. l--;
771. **if** (relation(\*l, \*mid, u) \* flag <= 0)
772. **break**;
773. s.erase(mid);
774. }
775. **while** (**true**) {
776. auto mid = it;
777. mid++;
778. **if** (mid == s.end())
779. **break**;
780. auto r = mid;
781. r++;
782. **if** (r == s.end())
783. **break**;
784. **if** (relation(u, \*mid, \*r) \* flag <= 0)
785. **break**;
786. s.erase(mid);
787. }
788. }
789. **void** insert(Point u) {
790. insert(su, u, 1), insert(sd, u, -1);
791. }
792. **int** check(Point u) {
793. **return** query(su, u, 1) && query(sd, u, -1);
794. }
795. };
797. Circle min\_circle\_cover(vector<Point> q) {
798. **int** n = q.size();
799. random\_shuffle(all(q));
800. Circle c = (Circle){q[0], 0};
801. auto get\_line = [](Point a, Point b) {  // 求中垂线
802. **return** Line({(a + b) / 2}, {(b - a).rotate(-pi / 2)});
803. };
804. auto get\_circle = [&](Point a, Point b, Point c) {
805. Line l = get\_line(a, b), r = get\_line(a, c);
806. Point p = l.intersect(r);
807. **return** (Circle){p, get\_dis(p, a)};
808. };
809. **for** (**int** i = 1; i < n; i++) {    // O(n)
810. **if** (cmp(c.r, get\_dis(c.p, q[i])) < 0) {
811. c = {q[i], 0};
812. **for** (**int** j = 0; j < i; j++) {
813. **if** (cmp(c.r, get\_dis(c.p, q[j])) < 0) {
814. c = {(q[i] + q[j]) / 2, get\_dis(q[i], q[j]) / 2};
815. **for** (**int** k = 0; k < j; k++) {
816. **if** (cmp(c.r, get\_dis(c.p, q[k])) < 0)
817. c = get\_circle(q[i], q[j], q[k]);
818. }
819. }
820. }
821. }
822. }
823. **return** c;
824. }

多面体欧拉定理



三维计算几何：

1. **const** **int** N = 1000 + 100;
2. **const** **int** inf = 1e9;
3. //const ll inf = 1e18;
4. **const** ll mod = 998244353;//1e9 + 7;
6. **const** **double** eps = 1e-14, pi = acos(-1);
7. **int** sign(**double** x) {
8. **return** x < -eps ? -1 : x > eps;
9. }
10. **int** cmp(**double** x, **double** y) {
11. **if** (fabs(x - y) < eps)
12. **return** 0;
13. **if** (x < y)
14. **return** -1;
15. **return** 1;
16. }
18. **struct** Point {
19. **double** x, y, z;
20. Point(**double** x = 0, **double** y = 0, **double** z = 0) : x(x), y(y), z(z) {}
21. **bool** operator<(**const** Point& B) **const** {
22. **if** (cmp(x, B.x))
23. **return** x < B.x;
24. **if** (cmp(y, B.y))
25. **return** y < B.y;
26. **return** z < B.z;
27. }
28. **bool** operator==(**const** Point& B) **const** {
29. **return** !cmp(x, B.x) && !cmp(y, B.y) && !cmp(z, B.z);
30. }
31. Point operator+(**const** Point& B) **const** {
32. **return** Point(x + B.x, y + B.y, z + B.z);
33. }
34. Point operator-(**const** Point& B) **const** {
35. **return** Point(x - B.x, y - B.y, z - B.z);
36. }
37. Point operator\*(**const** **double** a) **const** {
38. **return** Point(x \* a, y \* a, z \* a);
39. }
40. Point operator/(**const** **double** a) **const** {
41. **return** Point(x / a, y / a, z / a);
42. }
43. **double** operator\*(**const** Point& B) **const** {
44. **return** x \* B.x + y \* B.y + z \* B.z;
45. }
46. Point operator^(**const** Point& B) **const** {
47. **return** {y \* B.z - B.y \* z, z \* B.x - x \* B.z, x \* B.y - y \* B.x};
48. }
49. **double** length() {
50. **return** sqrt(x \* x + y \* y + z \* z);
51. }
52. **double** rand\_eps() {
53. **return** ((**double**)rand() / RAND\_MAX - 0.5) \* eps;
54. }
55. Point trunc(**double** len) {
56. len /= length();
57. x \*= len, y \*= len, z \*= len;
58. **return** \***this**;
59. }
60. **void** shake() {  // 微小扰动, 防止出现四点共面的情况
61. x += rand\_eps(), y += rand\_eps(), z += rand\_eps();
62. }
63. };
65. **struct** Line {
66. Point p, v;
67. Line() {}
68. Line(**const** Point& A, **const** Point& B) : p(A), v(B) {}
69. Point foot\_point(Point a) {     // 垂足
70. **return** p + v \* ((v \* (a - p)) / (v \* v));
71. }
72. **double** distance(**const** Point& a) {   // 点到直线的距离
73. **return** fabs((v ^ (a - p)).length() / v.length());
74. }
75. };
77. // 平面类
78. Point q[N];
79. **struct** Plane {
80. **int** v[3];
81. Point norm() {  // 平面法向量
82. **return** (q[v[1]] - q[v[0]]) ^ (q[v[2]] - q[v[0]]);
83. }
84. **double** area() {  // 面积
85. **return** norm().length() / 2;
86. }
87. **bool** above(Point a) {  // 点a是否在平面的上方
88. **return** ((a - q[v[0]]) \* norm()) >= 0;
89. }
90. };
92. Plane plane[N], np[N];  // 通过微小扰动, 防止四点共面
93. **int** n, m;
94. **bool** g[N][N];           // 某条边是否是分界线
95. **void** get\_convex\_3d() {  // O(n^2)
96. plane[m++] = {0, 1, 2}, plane[m++] = {2, 1, 0};
97. **for** (**int** i = 3; i < n; i++) {
98. **int** cnt = 0;
99. **for** (**int** j = 0; j < m; j++) {
100. **bool** t = plane[j].above(q[i]);
101. **if** (!t)
102. np[cnt++] = plane[j];
103. **for** (**int** k = 0; k < 3; k++)
104. g[plane[j].v[k]][plane[j].v[(k + 1) % 3]] = t;
105. }
106. **for** (**int** j = 0; j < m; j++) {
107. **for** (**int** k = 0; k < 3; k++) {
108. **int** a = plane[j].v[k], b = plane[j].v[(k + 1) % 3];
109. **if** (g[a][b] && !g[b][a])
110. np[cnt++] = {a, b, i};
111. }
112. }
113. m = cnt;
114. **for** (**int** j = 0; j < m; j++)
115. plane[j] = np[j];
116. }
117. }
118. **signed** main() {
119. cin >> n;
120. **for** (**int** i = 0; i < n; i++) {
121. cin >> q[i].x >> q[i].y >> q[i].z;
122. q[i].shake();
123. }
124. get\_convex\_3d();
125. **double** ans = 0;
126. **for** (**int** i = 0; i < m; i++)
127. ans += plane[i].area();
128. cout << fixed << setprecision(6) << ans << endl;
129. }
131. Line planeIntersection(Point a, Point v1, Point b, Point v2) {  // 平面AB的交线 (已知平面上一点及其法向量)
132. Point e = (v1 ^ v2), u = (v1 ^ e);
133. **double** d = v2 \* u;
134. **if** (!sign(d))
135. **return** Line(Point(0, 0), Point(0, 0));
136. Point q = a + u \* ((v2 \* (b - a)) / d);
137. **return** Line(q, e);
138. }
140. //经纬度转化为空间坐标
141. **double** torad(**double** a) {   // 角度转化为弧度
142. **return** a / 180 \* pi;
143. }
144. Point getPoint(**double** R, **double** a, **double** b) {  // a经度,b维度,均为角度
145. a = torad(a), b = torad(b);
146. **return** Point(R \* cos(a) \* cos(b), R \* cos(a) \* sin(b), R \* sin(a));
147. }
149. //两点球面距离
150. **double** ball\_dist(**double** R, Point a, Point b) {
151. **double** d = (a - b).length();
152. **return** 2 \* asin(d / 2 / R) \* R;
153. }
155. //直线与球交点
156. **int** lineInterBall(Line L, Point O, **double** R, Point& p1, Point& p2) {
157. Point foot = L.foot\_point(O);
158. **double** distOF = (O - foot).length();
159. **if** (cmp(distOF, R) > 0)
160. **return** 0;
161. **else** **if** (!cmp(distOF, R)) {
162. p1 = p2 = foot;
163. **return** 1;
164. }
165. Point delta = L.v.trunc(sqrt(R \* R - distOF \* distOF));
166. p1 = foot + delta, p2 = foot - delta;
167. **return** 2;
168. }

# 十八、高精度（加减法，无负数）

1. **struct** Bignum {
2. vector<**int**> val;
3. **int** len;
4. Bignum(){};
5. Bignum(string s) {
6. len = s.length();
7. val.resize(len);
8. **for** (**int** i = 0; i < len; i++) {
9. assert(s[i] >= '0' && s[i] <= '9');
10. val[i] = s[i] - '0';
11. }
12. reverse(all(val));
13. }
14. **bool** operator <(Bignum b) {
15. **if** (len < b.len) **return** 1;
16. **if** (len > b.len) **return** 0;
17. **for** (**int** i = len - 1; i >= 0; i--) {
18. **if** (val[i] != b.val[i]) **return** val[i] < b.val[i];
19. }
20. **return** 0;
21. }
22. **bool** operator ==(Bignum b) {
23. **if** (len != b.len) **return** 0;
24. **for** (**int** i = len - 1; i >= 0; i--) {
25. **if** (val[i] != b.val[i]) **return** 0;
26. }
27. **return** 1;
28. }
29. Bignum operator +(Bignum b) {
30. Bignum res;
31. **int** n = max(len, b.len);
32. **int** carry = 0;
34. res.val.resize(len);
35. val.resize(len);
36. b.val.resize(len);
38. **for** (**int** i = 0; i < n; i++) {
39. **int** nw = val[i] + b.val[i] + carry;
40. res.val[i] = nw % 10;
41. carry = nw / 10;
42. }
43. **if** (carry) res.val.pb(carry);
44. res.len = res.val.size();
45. **return** res;
46. }
47. Bignum operator -(Bignum b) {
48. assert(b < \***this** || b == \***this**);
49. Bignum res;
50. **int** n = max(len, b.len);
51. **int** carry = 0;
53. res.val.resize(len);
54. val.resize(len);
55. b.val.resize(len);
57. **for** (**int** i = 0; i < n; i++) {
58. **int** nw = val[i] - b.val[i] - carry;
59. **if** (nw < 0) {
60. carry = 1;
61. res.val[i] = nw + 10;
62. } **else** {
63. carry = 0;
64. res.val[i] = nw;
65. }
66. }
67. assert(!carry);
68. **while** (!res.val.empty() && res.val.back() == 0)res.val.pop\_back();
69. res.len = res.val.size();
70. **return** res;
71. }
72. string to\_string() {
73. string res = "";
74. **if** (len == 0) {
75. res += '0';
76. **return** res;
77. }
78. **for** (**int** i = len - 1; i >= 0; i--) {
79. res += (**char**)(val[i] + '0');
80. }
81. **return** res;
82. }
83. };

# 十九、python模板

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))
10. ##n = int(input())
11. ##a = list(map(int, input().split()))
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("")
24. ##alt3注释,alt4取消
25. ##F1看帮助文档
27. ##双端队列写bfs!!
28. ##n, m, sx, sy = map(int, input().strip().split())
29. ##
30. ##dis = [[-1] \* (m + 1) for i in range(n + 1)] ##多维数组酱紫开，深拷贝
31. ##dis[sx][sy] = 0
32. ##que = deque()
33. ##
34. ##dx = [-2, -2, -1, -1, 1, 1, 2, 2]
35. ##dy = [-1, 1, -2, 2, -2, 2, -1, 1]
36. ##
37. ##que.append([sx, sy])
38. ##
39. ##while que:
40. ##    x, y = que.popleft()
41. ##    for d in range(0, 8):
42. ##        nx, ny = x + dx[d], y + dy[d]
43. ##        if nx < 1 or nx > n or ny < 1 or ny > m:
44. ##            continue
45. ##        if dis[nx][ny] == -1:
46. ##            dis[nx][ny] = dis[x][y] + 1
47. ##            que.append([nx, ny])
48. ##
49. ##for i in range(1, n + 1):
50. ##    for j in range(1, m + 1):
51. ##        print(dis[i][j], end = ' ')
52. ##    print("")
54. ##堆的用法& dijkstra最短路
55. ##n, m, s = map(int, input().strip().split())
56. ##adj = [[] for i in range(0, n + 5)]
57. ##for i in range(0, m):
58. ##    x, y, z = map(int, input().strip().split())
59. ##    adj[x].append([y, z])
60. ##
61. ##dis = [(int)(1e18)] \* (n + 1)
62. ##dis[s] = 0
63. ##pq = [[0, s]]
64. ##heapq.heapify(pq)
65. ##while pq:
66. ##    d, nw = heapq.heappop(pq)
67. ##    if d != dis[nw]:
68. ##        continue
69. ##    for v, w in adj[nw]:
70. ##        if dis[v] > d + w:
71. ##            dis[v] = d + w
72. ##            heapq.heappush(pq, [d + w, v])
73. ##for i in range(1, n + 1):
74. ##    print(dis[i], end = ' ')
76. ##从某个库调用方法好一点！码出‘heapq.’后，按tab会出自动补全，good！
77. ##线段树！！
79. n, m = map(int, input().strip().split())
80. a = list(map(int, input().split()))
81. list.insert(a, 0, 0)
83. seg1 = [0] \* (4 \* n + 5)
84. laz1 = [0] \* (4 \* n + 5)
85. seg2 = [0] \* (4 \* n + 5)
86. laz2 = [0] \* (4 \* n + 5)
88. **def** ls(k:int):
89. **return** k << 1
90. **def** rs(k:int):
91. **return** k << 1 | 1
92. **def** push\_up(seg, k:int):
93. seg[k] = seg[ls(k)] + seg[rs(k)]
94. **def** push\_down(seg, laz, l, r, k:int):
95. **if** laz[k]:
96. mid = (l + r) >> 1
97. seg[ls(k)] += laz[k] \* (mid - l + 1)
98. seg[rs(k)] += laz[k] \* (r - mid)
99. laz[ls(k)] += laz[k]
100. laz[rs(k)] += laz[k]
101. laz[k] = 0
102. **def** ask(seg, laz, L, R, l = 1, r = n, k = 1):
103. **if** l > R **or** r < L:
104. **return** 0
105. **if** L <= l **and** r <= R:
106. **return** seg[k]
107. push\_down(seg, laz, l, r, k)
108. mid = (l + r) >> 1
109. **return** ask(seg, laz, L, R, l, mid, ls(k)) + ask(seg, laz, L, R, mid + 1, r, rs(k))
110. **def** upd(seg, laz, L, R, val, l = 1, r = n, k = 1):
111. **if** l > R **or** r < L:
112. **return**
113. **if** L <= l **and** r <= R:
114. seg[k] += val
115. laz[k] += val
116. **return**
117. push\_down(seg, laz, l, r, k)
118. mid = (l + r) >> 1
119. upd(seg, laz, L, R, val, l, mid, ls(k))
120. upd(seg, laz, L, R, val, mid + 1, r, rs(k))
121. push\_up(seg, k)
122. **return**
123. **def** build(seg, laz, l = 1, r = n, k = 1):
124. laz[k] = 0
125. **if** l == r:
126. seg[k]= a[l]
127. **return**
128. mid = (l + r) >> 1
129. build(seg, laz, l, mid, ls(k))
130. build(seg, laz, mid + 1, r, rs(k))
131. push\_up(seg, k)
132. **return**

135. build(seg1, laz1)
136. ##for \_ in range(0, m):
137. ##    opt = list(map(int, input().split()))
138. ##    if opt[0] == 1:
139. ##        upd(seg1, laz1, opt[1], opt[2], opt[3])
140. ##    else:
141. ##        print(ask(seg1, laz1, opt[1], opt[1]))
142. build(seg2, laz2)
143. **for** \_ **in** range(0, m):
144. opt = list(map(int, input().split()))
145. **if** opt[0] == 1:
146. upd(seg2, laz2, opt[1], opt[1], opt[2])
147. **else**:
148. **print**(ask(seg2, laz2, opt[1], opt[2]))

# 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！