ACM-ICPC 手册 (Julian 队特供)

ACM-ICPC 手册 (Julian 队特供)

> 改编自CSP-S 2020 考前总结

环境配置

使用 Obsidian 配色, 当前行黑色高亮, 字体默认 Consolas. 取消 使用Tab字符 选项, Tab位置 改为 2 (或你们喜欢的), 在 编译器选项\代码生成/优化\连接器\产生调试信息 中将 No 改为 Yes.

在编译选项中要打的所有命令为:

```
-std=c++11 -Wl,--stack=512000000 -Wall -Wconversion -Wextra -O2 -Wsign-compare
```

缺省源相关

头文件

```cpp
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <string>
#include <vector>
//<ctime> 库中和时间有关的函数都会影响后期申诉，所以不在要提交的答案代码中写
```

IO 优化

```cpp
inline unsigned RD() {   // 自然数
  unsigned intmp = 0;
  char rdch(getchar());
  while (rdch < '0' || rdch > '9') rdch = getchar();
  while (rdch >= '0' && rdch <= '9') intmp = intmp * 10 + rdch - '0', rdch = getchar();
  return intmp;
}
inline int RDsg() {    // 整数
  int rdtp(0), rdsg(1);
  char rdch(getchar());
  while ((rdch < '0' || rdch > '9') && (rdch != '-')) rdch = getchar();
  if (rdch == '-') rdsg = -1, rdch = getchar();
```

```
13    while (rdch >= '0' && rdch <= '9') rdtp = rdtp * 10 + rdch - '0', rdch =
   getchar();
14    return rdtp * rdsg;
15  }
16  inline void PR(long long Prtmp, bool SoE) {
17    unsigned long long Prstk(0), Prlen(0);
18    if (Prtmp < 0) putchar('-'), Prtmp = -Prtmp;
19    do {
20      Prstk = Prstk * 10 + Prtmp % 10, Prtmp /= 10, ++Prlen;
21    } while (Prtmp);
22    do {
23      putchar(Prstk % 10 + '0');
24      Prstk /= 10;
25      --Prlen;
26    } while (Prlen);
27    if (SoE) putchar('\n');
28    else putchar(' ');
29    return;
30  }
```

对拍

```
1  unsigned random(unsigned l, unsigned r) {return (rand() % (r - l + 1)) + l;}
2  int main() {  // random.cpp
3    freopen("balabala.in", "w", stdout);
4    ......
5    return 0;
6  }
```

```
1  int main() {
2    n = howManyTimesDoYouWant;
3    for (register unsigned i(1); i <= n; ++i) {
4      system("random.exe"), system("balabala_my.exe"),
   system("balabala_std.exe");
5      if(system("fc balabala_my.out balabala_std.out")) break;
6    }
7    return Wild_Donkey;
8  }
```

数据结构

并查集

```
1   int Fnd(const int &x) {
2     int x_tmp(x);
3     while (x_tmp!=Fthr[x_tmp]) x_tmp = Fthr[x_tmp];
4     Fthr[x] = x_tmp;//路径压缩
5     return x_tmp;
6   }
7   void Add(const int &x, const int &y) {
8     Fthr[Fnd(x)] = Fthr[y];
9     return;
10  }
```

St 表

```
for (register int i(1); i <= n; ++i) St[0][i] = RD();
Log2[1] = 0;
for (register int i(2); i <= n; ++i) {
  Log2[i] = Log2[i - 1];
  if(i >= 1 << (Log2[i - 1] + 1)) ++Log2[i];
}
void Bld() {
  for (register int i(1); i <= Log2[n]; ++i)
    for (register int j(1); j + (1 << i) <= n + 1; ++j)
      St[i][j] = max(St[i - 1][j], St[i - 1][j + (1 << (i - 1))]);
  return;
}
int Fnd () {
  int len = Log2[B - A + 1];
  return max(St[len][A], St[len][B - (1 << len) + 1]);
}
```

线段树

```
struct Node {
  Node *Ls, *Rs;
  long long Val, Tag, L, R;
} N[200005], *cntn(N);
long long a[100005];
int A, B, C, n, m, k, DWt;
void Udt(Node *x) {
  if(x->L == x->R) return;
  x->Val = x->Ls->Val + x->Rs->Val;
  return;
}
void Dld(Node *x) {
  if(!(x->Tag)) {
    return;
  }
  if(!(x->L == x->R)) {
    x->Ls->Tag += x->Tag;
    x->Rs->Tag += x->Tag;
    x->Ls->Val += x->Tag * (x->Ls->R - x->Ls->L + 1);
    x->Rs->Val += x->Tag * (x->Rs->R - x->Rs->L + 1);
  }
  x->Tag = 0;
  return;
}
void Chg(Node *x) {
  if(OtRg(x)) {
    return;
  }
  if(InRg(x)) {
    x->Tag += C;
    x->Val += C * (x->R - x->L + 1);
    return;
  }
  Dld(x);//就是这句忘写了qaq
  Chg(x->Ls);
```

```
36    Chg(x->Rs);
37    Udt(x);
38    return;
39  }
40  long long Fnd(Node *x) {//不开 long long 见祖宗
41    if(OtRg(x)) {
42      return 0;
43    }
44    if(InRg(x)) {
45      return x->Val;
46    }
47    Dld(x);
48    long long tmp (Fnd(x->Ls));
49    return tmp + (Fnd(x->Rs));
50  }
```

树状数组

```
1   unsigned int m, n, Dw;
2   int A, B, a[500005], T[500005];
3   inline unsigned int Lb(const int &x) { return x & ((~x) + 1); }
4   void Chg() {
5     for (register unsigned int i(A); i <= n; i = i + Lb(i)) T[i] += B;
6     return;
7   }
8   int Qry(const int &x) {
9     int y(0);
10    for (register unsigned int i(x); i; i -= Lb(i)) y += T[i];
11    return y;
12  }
13  int main() {
14    n = RD(), m = RD(), memset(T, 0, sizeof(T));
15    for (register unsigned int i(1); i <= n; ++i) a[i] = RD();
16    for (register unsigned int i(1); i <= n; ++i) {
17      T[i] = a[i];
18      for (register unsigned int j(Lb(i) >> 1); j; j = j >> 1) T[i] += T[i -
    j];
19    }
20    for (register unsigned int i(1); i <= m; ++i) {
21      Dw = RD(), A = RD(), B = RD();
22      if(Dw & 1) Chg();
23      else printf("%d\n", Qry(B) - Qry(A - 1));
24    }
25    return 0;
26  }
```

分块 (蒲公英)

```
1   int main() {
2     n = RD(), m = RD(), memset(Ap, 0, sizeof(Ap)), Rg = max(int(sqrt(n)), 1);
    //确定Rg
3     NmR = (n + Rg - 1) / Rg;      //推得NmR
4     for (register int i(1); i <= n; ++i) a[i] = RD(), b[i] = a[i];  //创建 a[]
    副本
5     sort(b + 1, b + n + 1);
```

```
 6    for (register int i(1); i <= n; ++i) if (b[i] != b[i - 1]) Ar[++Cnta] =
      b[i];    // Ar 存严格次序中第 k 小的数
 7    for (register int i(1); i <= n; ++i) a[i] = lower_bound(Ar + 1, Ar + Cnta
      + 1, a[i]) - Ar;    //离散化，将每个a[i]变成小于等于n的数
 8    for (register int i(1); i < NmR; ++i) {    //处理Ap[][]
 9      for (register int j(Rg * (i - 1) + 1); j <= Rg * i; ++j) ++Ap[i][a[j]];
10      for (register int j(1); j <= Cnta; ++j) Ap[i + 1][j] = Ap[i][j];//继承给
      下一块
11    }//最后一行
12    for (register int i(Rg * (NmR - 1) + 1); i <= n; ++i) ++Ap[NmR][a[i]];    //
      最后一行特殊处理
13    for (register int i(1); i < NmR; ++i) {    //处理长度为 1 块的区间的 f[][]
14      Tmp = 0;
15      for (register int j(Rg * (i - 1) + 1); j <= Rg * i;
16          ++j) {    //枚举每一个出现过的数字
17        if (Ap[i][Tmp] - Ap[i - 1][Tmp] <= Ap[i][a[j]] - Ap[i - 1][a[j]]) {
18          if (Ap[i][Tmp] - Ap[i - 1][Tmp] == Ap[i][a[j]] - Ap[i - 1][a[j]]) {
19            if (Tmp > a[j]) Tmp = a[j];
20          } else  Tmp = a[j];
21        }
22      }
23      f[i][i] = Tmp;
24    }
25    Tmp = 0;
26    for (register int i(Rg * (NmR - 1) + 1); i <= n; ++i) {    //最后一行特殊处理
27      if (Ap[NmR][Tmp] - Ap[NmR - 1][Tmp] <= Ap[NmR][a[i]] - Ap[NmR - 1]
      [a[i]]) {
28        if (Ap[NmR][Tmp] - Ap[NmR - 1][Tmp] == Ap[NmR][a[i]] - Ap[NmR - 1]
      [a[i]]) {
29          if (Tmp > a[i]) Tmp = a[i];
30        } else Tmp = a[i];
31      }
32      f[NmR][NmR] = Tmp;
33    }
34    for (register int i(1); i <= NmR; ++i) {
35      for (register int j(i + 1); j <= NmR; ++j) {    //处理全部f[][]
36        if (f[i][j - 1] == f[j][j]) {                //共同众数无需处理
37          f[i][j] = f[j][j];
38        } else {
39          Tmp = f[i][j - 1];
40          for (register int k(Rg * (j - 1) + 1); k <= min(Rg * j, n); ++k) {
      //枚举出现过的数字
41            if (Ap[j][Tmp] - Ap[i - 1][Tmp] <= Ap[j][a[k]] - Ap[i - 1][a[k]])
      {
42              if (Ap[j][Tmp] - Ap[i - 1][Tmp] == Ap[j][a[k]] - Ap[i - 1]
      [a[k]]) {
43                if (Tmp > a[k]) Tmp = a[k];    //数字小的优先
44              } else Tmp = a[k];    //更新众数
45            }
46          }
47          f[i][j] = Tmp;    //众数以确定
48        }
49      }
50    }
51    for (register int i(1); i <= m; ++i) {    //处理询问
52      L = (RD() + Lst - 1) % n + 1, R = (RD() + Lst - 1) % n + 1;    //区间生成(强
      制在线)
53      if (L > R) swap(L, R); //判左大右小
```

```
54        Lr = (L + Rg - 1) / Rg + 1, Rr = R / Rg;      //处理包含的最左块和最右块
55      if (Lr > Rr) {  //整块不存在
56        for (register int j(L); j <= R; ++j) Tmpp[a[j]] = 0;    //直接朴素，清空
   计数器(下同)
57        for (register int j(L); j <= R; ++j) ++Tmpp[a[j]];
58        Tmp = 0;
59        for (register int j(L); j <= R; ++j) {
60          if (Tmpp[Tmp] <= Tmpp[a[j]]) {
61            if (Tmpp[Tmp] == Tmpp[a[j]]) {
62              if (Tmp > a[j]) Tmp = a[j];
63            } else Tmp = a[j];
64          }
65        }
66        Lst = Tmp;
67      } else {             //有整块
68        Tmp = f[Lr][Rr];  //先和判整块众数出现次数比较
69        Tmpp[Tmp] = 0;     //别忘了这里
70        for (register int j(L); j <= Rg * (Lr - 1); ++j) Tmpp[a[j]] = 0;   //掐
   头
71        for (register int j(Rg * Rr + 1); j <= R; ++j) Tmpp[a[j]] = 0; //去尾
72        for (register int j(L); j <= Rg * (Lr - 1); ++j) ++Tmpp[a[j]];
73        for (register int j(Rg * Rr + 1); j <= R; ++j) ++Tmpp[a[j]];
74        for (register int j(L); j <= Rg * (Lr - 1); ++j) {  //开始迭代
75          if (Tmpp[Tmp] + Ap[Rr][Tmp] - Ap[Lr - 1][Tmp] <=
76              Tmpp[a[j]] + Ap[Rr][a[j]] -
77                Ap[Lr - 1][a[j]]) {   //当前数字出现次数和当前已知众数出现次数
78            if (Tmpp[Tmp] + Ap[Rr][Tmp] - Ap[Lr - 1][Tmp] ==
79                Tmpp[a[j]] + Ap[Rr][a[j]] - Ap[Lr - 1][a[j]]) {
80              if (Tmp > a[j]) Tmp = a[j];
81            } else Tmp = a[j];
82          }
83        }
84        for (register int j(Rg * Rr + 1); j <= R; ++j) {   //尾操作同头
85          if (Tmpp[Tmp] + Ap[Rr][Tmp] - Ap[Lr - 1][Tmp] <= Tmpp[a[j]] + Ap[Rr]
   [a[j]] - Ap[Lr - 1][a[j]]) {
86            if (Tmpp[Tmp] + Ap[Rr][Tmp] - Ap[Lr - 1][Tmp] == Tmpp[a[j]] +
   Ap[Rr][a[j]] - Ap[Lr - 1][a[j]]) {
87              if (Tmp > a[j]) Tmp = a[j];
88            } else Tmp = a[j];
89          }
90        }
91        Lst = Tmp;
92      }
93      Lst = Ar[Lst];   //离散化后的值转化为原始值
94      printf("%d\n", Lst);
95    }
96    return 0;
97  }
```

轻重链剖分

```
1  unsigned int m, n, Rot, cntd(0), DW;
2  unsigned int Mod, a[200005], C, A, B, yl, yr, yv;
3  struct Edge;
4  struct Node {
5    unsigned int Siz, Dep, Cntson, DFSr;
6    unsigned int Val;
```

```
   7    Node *Fa, *Top, *Hvy;
   8    Edge *Fst;
   9  } N[200005];
  10  struct Edge {
  11    Edge *Nxt;
  12    Node *To;
  13  } E[400005], *cnte(E);
  14  void Lnk(Node *x, Node *y) {
  15    (++cnte)->Nxt = x->Fst;
  16    x->Fst = cnte;
  17    cnte->To = y;
  18    return;
  19  }
  20  struct SgNode {
  21    unsigned int Val, Tag;
  22    SgNode *L, *R;
  23  } SgN[400005], *cntn(SgN);
  24  void SgBld(SgNode *x, const unsigned int &l, const unsigned int &r) {
  25    x->Tag = 0;
  26    if (l == r) {
  27      x->Val = a[l], x->L = x->R = NULL;
  28      return;
  29    }
  30    x->L = ++cntn;
  31    x->R = ++cntn;
  32    int mid((l + r) >> 1);
  33    SgBld(x->L, l, mid);
  34    SgBld(x->R, mid + 1, r);
  35    x->Val = x->L->Val + x->R->Val;
  36    return;
  37  }
  38  inline void PsDw(SgNode *x, const unsigned int &l, const unsigned int &r) {
  39    unsigned int mid((l + r) >> 1);
  40    if (mid < r) {
  41      x->L->Val += x->Tag * (mid - l + 1);
  42      x->R->Val += x->Tag * (r - mid);
  43      x->L->Tag += x->Tag;
  44      x->R->Tag += x->Tag;
  45    }
  46    x->Tag = 0;
  47    return;
  48  }
  49  inline void Udt(SgNode *x) {
  50    if (x->L) x->Val = (x->L->Val + x->R->Val) % Mod;
  51    return;
  52  }
  53  void SgChg(SgNode *x, const unsigned int &l, const unsigned int &r) {
  54    if (l == r) {
  55      x->Val += yv;
  56      return;
  57    }
  58    if ((l >= yl && r <= yr)) {
  59      x->Tag += yv, x->Val += (r - l + 1) * yv % Mod;
  60      return;
  61    }
  62    unsigned int mid((l + r) >> 1);
  63    if (x->Tag) PsDw(x, l, r);
  64    if (mid >= yl) SgChg(x->L, l, mid);
```

```
65      if (mid < yr) SgChg(x->R, mid + 1, r);
66    Udt(x);
67    return;
68  }
69  unsigned int SgQry(SgNode *x, const int &l, const int &r) {
70    if (l >= yl && r <= yr) return x->Val % Mod;
71    if (l == r) return Wild_Donkey;
72    if (x->Tag) PsDw(x, l, r);
73    unsigned int mid((l + r) >> 1), Tmp(0);
74    if (mid >= yl) Tmp += SgQry(x->L, l, mid);
75    if (mid < yr) Tmp += SgQry(x->R, mid + 1, r);
76    return Tmp % Mod;
77  }
78  void SonChg(Node *x) {
79    yl = x->DFSr, yr = x->DFSr + x->Siz - 1;
80    return SgChg(SgN, 1, n);
81  }
82  unsigned int SonQry(Node *x) {
83    yl = x->DFSr, yr = x->DFSr + x->Siz - 1;
84    return SgQry(SgN, 1, n);
85  }
86  void LnkChg(Node *x, Node *y) {
87    while (x->Top != y->Top) {
88      if (x->Top->Dep < y->Top->Dep) {
89        swap(x, y);
90      }
91      yl = x->Top->DFSr;
92      yr = x->DFSr;
93      SgChg(SgN, 1, n);
94      x = x->Top->Fa;
95    }
96    if (x->Dep < y->Dep) {
97      yl = x->DFSr;
98      yr = y->DFSr;
99      return SgChg(SgN, 1, n);
100   } else {
101     yl = y->DFSr;
102     yr = x->DFSr;
103     return SgChg(SgN, 1, n);
104   }
105   return;
106 }
107 unsigned int LnkQry(Node *x, Node *y) {
108   unsigned int Tmp(0);
109   while (x->Top != y->Top) {
110     if (x->Top->Dep < y->Top->Dep) swap(x, y);
111     yl = x->Top->DFSr, yr = x->DFSr, Tmp += SgQry(SgN, 1, n), x = x->Top-
    >Fa;
112   }
113   if (x->Dep < y->Dep) yl = x->DFSr, yr = y->DFSr, Tmp += SgQry(SgN, 1, n);
114   else yl = y->DFSr, yr = x->DFSr, Tmp += SgQry(SgN, 1, n);
115   return Tmp % Mod;
116 }
117 void Bld(Node *x) {
118   if (x->Fa) {
119     x->Dep = x->Fa->Dep + 1;
120   } else {
121     x->Dep = 1;
```

```
122      }
123    x->Siz = 1;
124    x->Cntson = 0;
125    Edge *Sid(x->Fst);
126    while (Sid) {
127      if (Sid->To != x->Fa) {
128        Sid->To->Fa = x, Bld(Sid->To);
129        if (!(x->Hvy)) x->Hvy = Sid->To;
130        else if (x->Hvy->Siz < Sid->To->Siz) x->Hvy = Sid->To;
131        x->Siz += Sid->To->Siz;
132        ++(x->Cntson);
133      }
134      Sid = Sid->Nxt;
135    }
136    return;
137  }
138  void DFS(Node *x) {
139    x->DFSr = (++cntd);
140    Edge *Sid(x->Fst);
141    if (x->Hvy) x->Hvy->Top = x->Top, DFS(x->Hvy);
142    else return;
143    while (Sid) {
144      if (Sid->To != x->Fa && Sid->To != x->Hvy)
145        Sid->To->Top = Sid->To, DFS(Sid->To);
146      Sid = Sid->Nxt;
147    }
148    return;
149  }
150  int main() {
151    n = RD(), m = RD(), Rot = RD(), Mod = RD();
152    for (register int i(1); i <= n; ++i) N[i].Val = RD() % Mod;
153    for (register int i(1); i < n; ++i) {
154      A = RD(), B = RD();
155      Lnk(N + A, N + B), Lnk(N + B, N + A);
156    }
157    Bld(N + Rot);
158    N[Rot].Top = N + Rot;
159    DFS(N + Rot);
160    for (register unsigned int i(1); i <= n; ++i) a[N[i].DFSr] = N[i].Val;
161    SgBld(SgN, 1, n);
162    for (register unsigned int i(1); i <= m; ++i) {
163      DW = RD(), A = RD();
164      switch (DW) {
165        case 1: {
166          B = RD();
167          yv = RD() % Mod;
168          LnkChg(N + A, N + B);
169          break;
170        }
171        case 2: {
172          B = RD();
173          printf("%u\n", LnkQry(N + A, N + B));
174          break;
175        }
176        case 3: {
177          yv = RD() % Mod;
178          SonChg(N + A);
179          break;
```

```
180          }
181        case 4: {
182          printf("%u\n", SonQry(N + A));
183          break;
184        }
185        default: {
186          printf("FYSNB\n");
187          break;
188        }
189      }
190    }
191    return 0;
192  }
```

可持久化数组

```
1   int m, n;
2   int a[1000005], A, B, C, D, Lst;
3   struct Node {
4     Node *L, *R;
5     int Val;
6   } N[20000005], *Vrsn[1000005], *Cntn(N);
7   void Bld(Node *x, unsigned int l, const unsigned int &r) {
8     if (l == r) {
9       x->Val = a[l];
10      return;
11    }
12    unsigned int m((l + r) >> 1);
13    Bld(x->L = ++Cntn, l, m);
14    Bld(x->R = ++Cntn, m + 1, r);
15    return;
16  }
17  void Chg(Node *x, Node *y, unsigned int l, const unsigned int &r) {
18    if (l == r) {
19      x->Val = D;
20      return;
21    }
22    unsigned int m = (l + r) >> 1;
23    if (C <= m) {                        //左边
24      x->R = y->R;                       //继承右儿子
25      Chg(x->L = ++Cntn, y->L, l, m);    //递归左儿子
26    } else {                             //右边
27      x->L = y->L;                       //继承左儿子
28      Chg(x->R = ++Cntn, y->R, m + 1, r);  //递归右儿子
29    }
30    return;
31  }
32  void Qry(Node *x, unsigned int l, const unsigned int &r) {
33    if (l == r) {
34      Lst = x->Val;
35      return;
36    }
37    unsigned int m = (l + r) >> 1;
38    if (C <= m) Qry(x->L, l, m);//左边，递归左儿子
39    else Qry(x->R, m + 1, r);//右边，递归右儿子
40    return;
41  }
```

```
42  int main() {
43    n = RD(), m = RD();
44    for (register int i(1); i <= n; ++i) a[i] = RD();
45    Bld(N, 1, n);
46    Vrsn[0] = N;
47    for (register int i(1); i <= m; ++i) {
48      A = RD(), B = RD(), C = RD();
49      if (B == 1) {
50        D = RD();
51        Vrsn[i] = ++Cntn;
52        Chg(Vrsn[i], Vrsn[A], 1, n);
53      } else {
54        Vrsn[i] = Vrsn[A];
55        Qry(Vrsn[i], 1, n);
56        printf("%d\n", Lst);
57      }
58    }
59    return 0;
60  }
```

主席树

```
 1  int a[200005], b[200005], Rkx[200005], A, B, C;
 2  unsigned int M, n, Cnta(0), Lst, Now;
 3  struct Node {
 4    Node *L, *R;
 5    unsigned int Val;
 6  } N[4000005], *Vrsn[200005], *Cntn(N);
 7  void Chg(Node *x, Node *y, unsigned int l, const unsigned int &r) {
 8    if (y) x->Val = y->Val + 1;
 9    else x->Val = 1;
10    if (l == r) return;
11    unsigned int m = (l + r) >> 1;
12    if (B <= m) {   //左边
13      if (y) {
14        x->R = y->R;                      //继承右儿子
15        Chg(x->L = ++Cntn, y->L, l, m);   //递归左儿子
16      } else {
17        x->R = NULL;
18        Chg(x->L = ++Cntn, NULL, l, m);   //递归左儿子
19      }
20    } else {   //右边
21      if (y) {
22        x->L = y->L;
23        Chg(x->R = ++Cntn, y->R, m + 1, r);   //递归右儿子
24      } else {
25        x->L = NULL;
26        Chg(x->R = ++Cntn, NULL, m + 1, r);   //递归右儿子
27      }                                       //继承左儿子
28    }
29    return;
30  }
31  void Qry(Node *x, Node *y, unsigned int l, const unsigned int &r) {
32    if (l == r) {   //边界
33      Lst = l;
34      return;
35    }
```

```
36     unsigned int m = (l + r) >> 1, Tmpx(0), Tmpy(0);
37     Node *Sonxl(NULL), *Sonxr(NULL), *Sonyl(NULL), *Sonyr(NULL);
38     if (x) {
39       if (x->L) Tmpx = x->L->Val, Sonxl = x->L;
40       if (x->R) Sonxr = x->R;
41     }
42     if (y) {
43       if (y->L) Tmpy = y->L->Val, Sonyl = y->L;
44       if (y->R) Sonyr = y->R;
45     }
46     if (C <= Tmpy - Tmpx) return Qry(Sonxl, Sonyl, l, m);//在左边，递归左儿子
47     C += Tmpx, C -= Tmpy;  //右边
48     return Qry(Sonxr, Sonyr, m + 1, r);  //递归右儿子
49   }
50   int main() {
51     n = RD(), M = RD();
52     memset(N, 0, sizeof(N));
53     for (register int i(1); i <= n; ++i) b[i] = a[i] = RD();
54     sort(b + 1, b + n + 1);
55     b[0] = 0x3f3f3f3f;
56     for (register int i(1); i <= n; ++i) if (b[i] != b[i - 1]) Rkx[++Cnta] =
     b[i];  // Rkx[i]为第i大的数为多少
57     Vrsn[0] = N;
58     for (register int i(1); i <= n; ++i) {
59       A = i;
60       B = lower_bound(Rkx + 1, Rkx + Cnta + 1, a[i]) - Rkx;
61       Chg(Vrsn[i] = ++Cntn, Vrsn[i - 1], 1, Cnta);
62     }
63     for (register int i(1); i <= M; ++i) {
64       A = RD(), B = RD(), C = RD();
65       Qry(Vrsn[A - 1], Vrsn[B], 1, Cnta);
66       printf("%d\n", Rkx[Lst]);
67     }
68     return 0;
69   }
```

Splay (强制在线, 数据加强版)

```
1    unsigned a[100005], b[100005], m, n, RealN(0), Cnt(0), C, D, t, Tmp(0);
2    bool Flg(0);
3    struct Node {
4      Node *Fa, *LS, *RS;
5      unsigned Value, Size, Count;
6    }N[1100005], *CntN(N), *Root(N);
7    Node *Build(register unsigned Le, register unsigned Ri, register Node
     *Father) {
8      if(Le ^ Ri) { // This Subtree is Bigger than 1
9        unsigned Mid((Le + Ri) >> 1);
10       Node *x(++CntN);
11       x->Count = b[Mid];
12       x->Size = b[Mid];
13       x->Value = a[Mid];
14       x->Fa = Father;
15       if(Le ^ Mid) x->LS = Build(Le, Mid - 1, x), x->Size += x->LS->Size;
16       x->RS = Build(Mid + 1, Ri, x);
17       x->Size += x->RS->Size;
18       return x;
```

```cpp
  }
  (++CntN)->Count = b[Le];// Single Point
  CntN->Size = b[Le];
  CntN->Value = a[Le];
  CntN->Fa = Father;
  return CntN;
}
inline void Rotate(register Node *x) {  // 绕父旋转
  if (x->Fa){
    Node *Father(x->Fa);                  // 暂存父亲
    x->Fa = Father->Fa;                   // 父亲连到爷爷上
    if(Father->Fa) {                      // Grandfather's Son (更新爷爷的儿子指
针)
      if(Father == Father->Fa->LS) Father->Fa->LS = x; // Left Son
      else Father->Fa->RS = x;            // Right Son
    }
    x->Size = x->Count;                   // x 的 Size 的一部分 (x->Size = x-
>LS->Size + x->RS->Size + x->Count)
    if(x == Father->LS) {                 // x is the Left Son, Zag(x->Fa)
      if(x->LS) x->Size += x->LS->Size;
      Father->LS = x->RS, x->RS = Father;
      if(Father->LS) Father->LS->Fa = Father;
    }
    else {                                // x is the Right Son, Zig(x->Fa)
      if(x->RS) x->Size += x->RS->Size;
      Father->RS = x->LS, x->LS = Father;
      if(Father->RS) Father->RS->Fa = Father;
    }
    Father->Fa = x/*父亲的新父亲是 x*/, Father->Size = Father->Count/*Father-
>Size 的一部分*/;
    if(Father->LS) Father->Size += Father->LS->Size; // 处理 Father 两个儿子
对 Father->Size 的贡献
    if(Father->RS) Father->Size += Father->RS->Size;
    x->Size += Father->Size;             // Father->Size 更新后才能更新 x-
>Size
  }
  return;
}
void Splay(Node *x) {
  if(x->Fa) {
    while (x->Fa->Fa) {
      if(x == x->Fa->LS) { // Boy
        if(x->Fa == x->Fa->Fa->LS) Rotate(x->Fa); // Boy & Father
        else Rotate(x);        // Boy & Mother
      }
      else {                   // Girl
        if(x->Fa == x->Fa->Fa->LS) Rotate(x);  // Girl & Father
        else Rotate(x->Fa);         // Girl & Mother
      }
    }
    Rotate(x);
  }
  Root = x;
  return;
}
void Insert(register Node *x, unsigned &y) {
  while (x->Value ^ y) {
    ++(x->Size);        // 作为加入元素的父节点，子树大小增加
```

```
 72        if(y < x->Value) {// 在左子树上
 73          if(x->LS) {      // 有左子树，往下走
 74            x = x->LS;
 75            continue;
 76          }
 77          else {           // 无左子树，建新节点
 78            x->LS = ++CntN;
 79            CntN->Fa = x;
 80            CntN->Value = y;
 81            CntN->Size = 1;
 82            CntN->Count = 1;
 83            return Splay(CntN);
 84          }
 85        }
 86        else {             // 右子树的情况同理
 87          if(x->RS) x = x->RS;
 88          else {
 89            x->RS = ++CntN;
 90            CntN->Fa = x;
 91            CntN->Value = y;
 92            CntN->Size = 1;
 93            CntN->Count = 1;
 94            return Splay(CntN);
 95          }
 96        }
 97      }
 98      ++(x->Count), ++x->Size;   // 原来就有对应节点
 99      Splay(x);                  // Splay 维护 BST 的深度复杂度
100      return;
101    }
102    void Delete(register Node *x, unsigned &y) {
103      while (x->Value ^ y) {
104        x = (y < x->Value) ? x->LS : x->RS;
105        if(!x) return;
106      }
107      Splay(x);
108      if(x->Count ^ 1) {       // Don't Need to Delete the Node
109        --(x->Count), --(x->Size);
110        return;
111      }
112      if(x->LS && x->RS) {     // Both Sons left
113        register Node *Son(x->LS);
114        while (Son->RS) Son = Son->RS;
115        x->LS->Fa = NULL/*Delete x*/, Splay(Son);// Let the biggest Node in (x-
       >LS) (the subtree) be the new root
116        Root->RS = x->RS, x->RS->Fa = Root; // The right son is still the right
       son
117        Root->Size = Root->Count + x->RS->Size;
118        if(Root->LS) Root->Size += Root->LS->Size;
119        return;
120      }
121      if(x->LS) x->LS->Fa = NULL, Root = x->LS; // x->LS is the new Root, x is
       The Biggest Number
122      if(x->RS) x->RS->Fa = NULL, Root = x->RS; // x->LS is the new Root, x is
       The Smallest Number
123      return;
124    }
125    void Value_Rank(register Node *x, unsigned &y, unsigned &Rank) {
```

```
126    while (x->Value ^ y) {   // Go Down
127      if(y < x->Value) {     // Go Left
128        if(x->LS) {
129          x = x->LS;
130          continue;
131        }
132        return;              // No more numbers smaller than y, Rank is the
rank
133      }
134      else {                 // Go Right
135        if(x->LS) Rank += x->LS->Size; // The Left Subtree numbers
136        Rank += x->Count;          // Mid Point numbers
137        if(x->RS) {
138          x = x->RS;
139          continue;
140        }
141        return;              // No more numbers bigger than y, Rank is the
rank
142      }
143    }
144    if(x->LS) Rank += x->LS->Size;// now, x->Value == y
145    return;
146  }
147  void Rank_Value(register Node *x, unsigned &y) {
148    while (x) {
149      if(x->LS) {
150        if(x->LS->Size < y) y -= x->LS->Size;//Not in the Left
151        else {             // In Left Subtree
152          x = x->LS;
153          continue;
154        }
155      }
156      if(y > x->Count) {  // In Right Subtree
157        y -= x->Count;
158        x = x->RS;
159        continue;
160      }
161      return Splay(x);    // Just Look for x
162    }
163  }
164  void Before(register Node *x, unsigned &y) {
165    while (x) {
166      if(y <= x->Value) {   // Go left
167        if(x->LS) {
168          x = x->LS;
169          continue;
170        }
171        while (x) {         // Go Up
172          if(x->Value < y) return Splay(x);
173          x = x->Fa;
174        }
175      }
176      else {                // Go right
177        if(x->RS) {
178          x = x->RS;
179          continue;
180        }
181        return Splay(x);    // Value[x] < Key
```

```cpp
        }
      }
    }
    void After(register Node *x, unsigned &y) {
      while (x) {
        if(y >= x->Value) {        // Go right
          if(x->RS) {
            x = x->RS;
            continue;
          }
          while (x) {               // Go Up
            if(x->Value > y) return Splay(x);
            x = x->Fa;
          }
        }
        else {                      // Go left
          if(x->LS) {
            x = x->LS;
            continue;
          }
          return Splay(x);
        }
      }
    }
    signed main() {
      register unsigned Ans(0);   // 记录
      n = RD();
      m = RD();
      a[0] = 0x7f3f3f3f;
      for (register unsigned i(1); i <= n; ++i) a[i] = RD();
      sort(a + 1, a + n + 1);
      for (register unsigned i(1); i <= n; ++i) {
        if(a[i] ^ a[i - 1]) b[++RealN] = 1, a[RealN] = a[i]; // A new number
        else ++b[RealN];          // Old number
      }
      a[++RealN] = 0x7f3f3f3f;
      b[RealN] = 1;
      Build(1, RealN, NULL);
      Root = N + 1;
      for (register unsigned i(1), A, B, Last(0); i <= m; ++i) {
        A = RD(), B = RD() ^ Last;
        switch(A) {
          case 1:{
            Insert(Root, B);
            break;
          }
          case 2:{
            Delete(Root, B);
            break;
          }
          case 3:{
            Last = 1;
            Value_Rank(Root, B, Last);
            Ans ^= Last;
            break;
          }
          case 4:{
            Rank_Value(Root, B);
```

```
240          Last = Root->Value;
241          Ans ^= Last;
242          break;
243        }
244        case 5:{
245          Before(Root, B);
246          Last = Root->Value;
247          Ans ^= Last;
248          break;
249        }
250        case 6:{
251          After(Root, B);
252          Last = Root->Value;
253          Ans ^= Last;
254          break;
255        }
256      }
257    }
258    printf("%u\n", Ans);
259    return Wild_Donkey;
260  }
```

Link/Cut Tree

```
1  unsigned a[10005], n, m, Cnt(0), Tmp(0), Mx;
2  bool flg(0);
3  char inch, List[155][75];
4  struct Node {
5    Node *Son[2], *Fa;
6    char Tag;
7    unsigned Value, Sum;
8  }N[100005], *Stack[100005];
9  inline void Update(Node *x) {
10   x->Sum = x->Value;
11   if(x->Son[0]) {
12     x->Sum ^= x->Son[0]->Sum;
13   }
14   if(x->Son[1]) {
15     x->Sum ^= x->Son[1]->Sum;
16   }
17   return;
18 }
19 inline void Push_Down(Node *x) {  // Push_Down the spliting tag
20   if(x->Tag) {
21     register Node *TmpSon(x->Son[0]);
22     x->Tag = 0, x->Son[0] = x->Son[1], x->Son[1] = TmpSon;
23     if(x->Son[0]) {
24       x->Son[0]->Tag ^= 1;
25     }
26     if(x->Son[1]) {
27       x->Son[1]->Tag ^= 1;
28     }
29   }
30 }
31 inline void Rotate(Node *x) {
32   register Node *Father(x->Fa);
33   x->Fa = Father->Fa; // x link to grandfather
```

```
34      if(Father->Fa) {
35        if(Father->Fa->Son[0] == Father) {
36          Father->Fa->Son[0] = x;   // grandfather link to x
37        }
38        if(Father->Fa->Son[1] == Father) {
39          Father->Fa->Son[1] = x;   // grandfather link to x
40        }
41      }
42      x->Sum = 0, Father->Fa = x;
43      if(Father->Son[0] == x) {
44        Father->Son[0] = x->Son[1];
45        if(Father->Son[0]) {
46          Father->Son[0]->Fa = Father;
47        }
48        x->Son[1] = Father;
49        if(x->Son[0]) {
50          x->Sum = x->Son[0]->Sum;
51        }
52      }
53      else {
54        Father->Son[1] = x->Son[0];
55        if(Father->Son[1]) {
56          Father->Son[1]->Fa = Father;
57        }
58        x->Son[0] = Father;
59        if(x->Son[1]) {
60          x->Sum = x->Son[1]->Sum;
61        }
62      }
63      Update(Father);
64      x->Sum ^= x->Value ^ Father->Sum;
65      return;
66    }
67    void Splay (Node *x) {
68      register unsigned Head(0);
69      while (x->Fa) {                                    // 父亲没到头
70        if(x->Fa->Son[0] == x || x->Fa->Son[1] == x) {       // x is the
      preferred-edge linked son (实边连接的儿子)
71          Stack[++Head] = x;
72          x = x->Fa;
73          continue;
74        }
75        break;
76      }
77      Push_Down(x);
78      if(Head) {
79        for (register unsigned i(Head); i > 0; --i) {//Must be sure there's no
      tags alone Root-x, and delete Root->Fa for a while
80          Push_Down(Stack[i]);
81        }
82        x = Stack[1];
83        while (x->Fa) {                                  // 父亲没到头
84          if(x->Fa->Son[0] == x || x->Fa->Son[1] == x) {  // x is the
      preferred-edge linked son (实边连接的儿子)
85            if (x->Fa->Fa) {
86              if (x->Fa->Fa->Son[0] == x->Fa || x->Fa->Fa->Son[1] == x->Fa) {
      // Father
```

```cpp
                Rotate((x->Fa->Son[0] == x)^(x->Fa->Fa->Son[0] == x->Fa) ? x :
x->Fa);
            }                          // End
        }
        Rotate(x);                  //最后一次旋转
      }
      else {
        break;
      }
    }
  }
  return;
}
void Access (Node *x) {      // Let x be the bottom of the chain where the
root at
  Splay(x), x->Son[1] = NULL, Update(x);          // Delete x's right son
  Node *Father(x->Fa);
  while (Father) {
    Splay(Father), Father->Son[1] = x; // Change the right son
    x = Father, Father = x->Fa, Update(x);      // Go up
  }
  return;
}
Node *Find_Root(Node *x) {  // Find the root
  Access(x), Splay(x), Push_Down(x);
  while (x->Son[0]) {
    x = x->Son[0], Push_Down(x);
  }
 Splay(x);
  return x;
}
int main() {
  n = RD(), m = RD();
  for (register unsigned i(1); i <= n; ++i) N[i].Value = RD();
  register unsigned A, B, C;
  for (register unsigned i(1); i <= m; ++i) {
    A = RD(), B = RD(), C = RD();
    switch (A) {
      case 0: { // Query
        Access(N + B), Splay(N + B), N[B].Tag ^= 1; // x 为根
        Access(N + C);      // y 和 x 为同一实链两端
        Splay(N + C);       // y 为所在实链的 Splay 的根
        printf("%u\n", N[C].Sum);
        break;
      }
      case 1: { // Link
        Access(N + B), Splay(N + B), N[B].Tag ^= 1;       // x 为根，也是所
在 Splay 的根
        if(Find_Root(N + C) != N + B) {// x, y 不连通，x 在 Fink_Root 时已经是
它所在 Splay 的根了，也是它原树根所在实链顶，左子树为空
          N[B].Fa = N + C;          // 父指针
        }
        break;
      }
      case 2: { // Cut
        Access(N + B), Splay(N + B), N[B].Tag ^= 1;
// x 为根，也是所在 Splay 的根
        if(Find_Root(N + C) == N + B) {            // x, y 连通
```

```
140          if(N[C].Fa == N + B && !(N[C].Son[0])) {// x 是 y 在 Splay 上的父亲,
     y 无左子树, 所以有直连边
141            N[C].Fa = N[B].Son[1] = NULL;          // 断边
142            Update(N + B);                          // 更新 x (y 的子树不变, 无
     需更新)
143          }
144        }
145        break;
146      }
147      case 3: { // Change
148        Splay(N + B);    // 转到根上
149        N[B].Value = C; // 改权值
150        break;
151      }
152    }
153  }
154  return Wild_Donkey;
155 }
```

DP

斜率优化

```
1  struct Ms {
2    long long C, T, SumC, SumT, f;
3  }M[5005]; // 任务属性
4  struct Hull {
5    long long x, y;
6    unsigned Ad;
7  }H[5005], *Now, Then; // 下凸壳
8  unsigned n, l(1), r(1);
9  long long S, Cst;
10 int main() {
11   n = RD(), S = RD(), M[0].SumT = S;
12   for (register unsigned i(1); i <= n; ++i) {
13     M[i].T = RD(), M[i].C = RD();
14     M[i].SumT = M[i - 1].SumT + M[i].T;
15     M[i].SumC = M[i - 1].SumC + M[i].C; //预处理
16   }
17   Cst = S * M[n].SumC;   // 截距中的一项常数
18   for (register unsigned i(1); i <= n; ++i) {
19     while (l < r && ((H[l + 1].y - H[l].y) < M[i].SumT * (H[l + 1].x -
     H[l].x))) ++l; // 弹出过气决策点
20     M[i].f = M[H[l].Ad].f + (M[i].SumC - M[H[l].Ad].SumC) * M[i].SumT + Cst
     - M[i].SumC * S; // 转移
21     Then.Ad = i, Then.x = M[i].SumC, Then.y = M[i].f;    // 求新点坐标
22     while (l < r && ((Then.y - H[r].y) * (H[r].x - H[r - 1].x) <= (H[r].y -
     H[r - 1].y) * (Then.x - H[r].x))) --r; // 维护下凸
23     H[++r] = Then;       // 入队
24   }
25   printf("%lld\n", M[n].f);
26   return Wild_Donkey;
27 }
```

斜率优化二分查找外挂

```
1  Hull *Binary (unsigned L, unsigned R, const long long &key) { // 在普通斜优的基
   础上的外挂
2    if(L == R) return H + L;
3    unsigned M((L + R) >> 1), M_ = M + 1;
4    if((H[M_].y - H[M].y) < key * (H[M_].x - H[M].x)) return Binary(M_, R,
   key); //Key too big
5    return Binary(L, M, key);
6  }
```

一维四边形不等式 (诗人小 G, 带路径)

```
1   #define Abs(x) ((x) > 0 ? (x) : -(x))
2   #define Do(x, y) (f[(x)] + Power(Abs(Sum[y] - Sum[x] - 1 - L), P))
3   inline void Clr() {
4     n = RD(), L = RD(), P = RD(), flg = 0, He = 1, Ta = 1;
5     Li[1].Adre = 0, Li[1].l = 1, Li[1].r = n, f[0] = 0, Sum[0] = 0; // 阶段 0
    是 0, 从 0 转移
6     char chtmp(getchar());
7     for (register unsigned i(1); i <= n; ++i) {
8       while (chtmp < 33 || chtmp > 127) chtmp = getchar();
9       a[i] = 0;
10      while (chtmp >= 33 && chtmp <= 127) Poem[i][a[i]++] = chtmp, chtmp =
    getchar();
11    }
12    return;
13  }
14  void Best(unsigned x) {
15    while (He < Ta && Do(Li[Ta].Adre, Li[Ta].l) >= Do(x, Li[Ta].l)) --Ta; //
    决策 x 对于区间起点表示的阶段更优，整个区间无用
16    if (Do(Li[Ta].Adre, Li[Ta].r) >= Do(x, Li[Ta].r)) Bin(x, Li[Ta].l,
    Li[Ta].r); // 决策 x 对于区间终点更优（至少一个阶段给 x）
17    else if (Li[Ta].r != n) ++Ta, Li[Ta].l = Li[Ta - 1].r + 1, Li[Ta].r = n,
    Li[Ta].Adre = x;
18    while (He < Ta && Li[He].r <= x) ++He; // 过时决策
19    Li[He].l = x + 1;
20    return;
21  }
22  void Best(unsigned x) {
23    while (He < Ta && Do(Li[Ta].Adre, Li[Ta].l) >= Do(x, Li[Ta].l)) --Ta; //
    决策 x 对于区间起点表示的阶段更优，整个区间无用
24    if (Do(Li[Ta].Adre, Li[Ta].r) >= Do(x, Li[Ta].r)) Bin(x, Li[Ta].l,
    Li[Ta].r); // 决策 x 对于区间终点更优（至少一个阶段给 x）
25    else if (Li[Ta].r != n) {
26      ++Ta;
27      Li[Ta].l = Li[Ta - 1].r + 1;
28      Li[Ta].r = n;
29      Li[Ta].Adre = x;
30    }
31    while (He < Ta && Li[He].r <= x) ++He; // 过时决策
32    Li[He].l = x + 1;
33    return;
34  }
35  inline void Bin(unsigned x /*新决策下标*/, unsigned le,
36              unsigned ri) {  // 区间内二分查找
```

```cpp
    if (le == ri) {                    // 新增一个区间
      Li[Ta].r = le - 1, Li[++Ta].l = le, Li[Ta].r = n, Li[Ta].Adre = x;
      return;
    }
    unsigned m((le + ri) >> 1);
    if (Do(x, m) <= Do(Li[Ta].Adre, m)) return Bin(x, le, m); // x 作为阶段 mid
的决策更优
    return Bin(x, m + 1, ri);
}
inline void Print() {Cnt = 0, Prt[0] = 0, Back(n);return;}
inline void Back(unsigned x) {
  if (Prt[x]) Back(Prt[x]);
  for (register unsigned i(Prt[x] + 1); i < x; ++i) {
    for (register short j(0); j < a[i]; ++j) putchar(Poem[i][j]);
    putchar(' ');
  }
  for (register short i(0); i < a[x]; ++i) putchar(Poem[x][i]);
  putchar('\n');
}
int main() {
  t = RD();
  for (register unsigned T(1); T <= t; ++T) {
    Clr();
    for (register unsigned i(1); i <= n; ++i) Sum[i] = Sum[i - 1] + a[i] +
1;
    for (register unsigned i(1); i < n; ++i) f[i] = Do(Li[He].Adre, i)/*从已
经求出的最优决策转移*/, Prt[i] = Li[He].Adre, Best(i);   // 更新数组 p
    f[n] = Do(Li[He].Adre, n);   // 从已经求出的最优决策转移
    Prt[n] = Li[He].Adre;
    if (f[n] > 1000000000000000000) printf("Too hard to arrange\n"); // 直接
溢出
    else printf("%lld\n", (long long)f[n]), Print();
    for (register short i(1); i <= 20; ++i) putchar('-');
    if (T < t) putchar('\n');
  }
  return Wild_Donkey;
}
```

二维四边形不等式 (邮局)

```cpp
for (register unsigned i(1); i <= n; ++i) a[i] = RD();
for (register unsigned i(1); i <= n; ++i) g[1][i] = 0;
for (register unsigned i(1); i <= n; ++i)
  for (register unsigned j(i + 1); j <= n; ++j)
    g[i][j] = g[i][j - 1] + a[j] - a[(i + j) >> 1]; // 预处理
memset(f, 0x3f, sizeof(f)), f[0][0] = 0;
for (register unsigned i(1); i <= n; ++i) {
  Dec[i][min(i, m) + 1] = 0x3f3f3f3f; // 对于本轮DP, Dec[i][min(i, m) + 1] 是
状态 (i, min(i, m)) 可行决策的右边界
  for (register unsigned j(min(i, m)); j >= 1; --j) {
    unsigned Mxn(min(i - 1, Dec[i][j + 1]));   // 右边界
    for (register unsigned k(Dec[i - 1][j])/*左边界*/; k <= Mxn; ++k) {
      if(f[k][j - 1] + g[k + 1][i] < f[i][j]) {
        f[i][j] = f[k][j - 1] + g[k + 1][i];
        Dec[i][j] = k;
      }
    }
  }
```

```
17      }
18      Dec[i][min(i, m) + 1] = 0;  // 对于下一轮，Dec[i][min(i, m) + 1] 是状态 (i +
    1, min(i, m)) 的左边界
19    }
```

图论

邻接表

```
 1  struct Edge;
 2  struct Node {
 3    Edge *Fst;
 4    int DFSr;
 5  }N[10005];
 6  struct Edge {
 7    Node *To;
 8    Edge *Nxt;
 9  }E[10005], *cnte(E);
10  void Lnk(const int &x, const int &y) {
11    (++cnte)->To = N + y;
12    cnte->Nxt = N[x].Fst;
13    N[x].Fst = cnte;
14    return;
15  }
16  void DFS(Node *x) {
17    x->DFSr = ++Dcnt;
18    Edge *Sid(x->Fst);
19    while (Sid) {
20      if(!Sid->To->DFSr) DFS(Sid->To);
21      Sid = Sid->Nxt;
22    }
23    return;
24  }
```

倍增 LCA (远古代码, 码风太嫩)

```
 1  struct Side {int to,next;};
 2  void LOG() {
 3    for(int i=1;i<=N;i++) LG[i]=LG[i-1]+(1<<LG[i-1]==i);//预先算出log2(i)+1的
    值，用的时候直接调用就可以了，如果1左移log(i-1)+1等于i,说明log(i)就等于log(i-1)+1
 4    return;
 5  }
 6  Side Sd[1000005];
 7  void BT(int a,int b) {
 8    Sd[++At].to=b, Sd[At].next=Fst[a], Fst[a]=At;
 9    return; }
10  void DFS(int at,int ft) {
11    Dp[at]=Dp[ft]+1;//深度比父亲大一
12    Tr[at][0]=ft;//往上走2^0(1)位就是父亲
13    int sd=Fst[at];
14    while(Sd[sd].to>0) {//深搜儿子
15      if(Sd[sd].to!=ft) DFS(Sd[sd].to,at);
16      sd=Sd[sd].next;
17    }
18    return;
```

```
19  }
20  int LCA(int a,int b) {
21    if(Dp[a]>Dp[b]) swap(a,b);
22    if(Dp[a]<Dp[b]) for(int i=LG[Dp[b]-Dp[a]]-1;i>=0;i--) if(Dp[b]-
    (1<<i)>=Dp[a]) b=Tr[b][i];//能跳则跳
23    if(a==b) return b;
24    else for(int i=LG[Dp[a]]-1;i>=0;i--) if(Tr[a][i]!=Tr[b][i]) a=Tr[a][i],
    b=Tr[b][i]; //走一遍之后,a,b差一步相遇,则他们的共同的父亲就是LCA,跳后a,b未相遇,则跳
25    return Tr[a][0];
26  }
27  int main() {
28    N=read(), M=read(), S=read(), Dp[S]=0;
29    memset(Sd,0,sizeof(Sd)), memset(Tr,0,sizeof(Tr)), memset(Dp,0,sizeof(Dp));
30    Dp[0]=-1, LOG();
31    for(int i=1;i<N;i++) X=read(), Y=read(), BT(X,Y), BT(Y,X);
32    DFS(S,0);//预处理深度和倍增数组
33    for(int i=1;i<=LG[N]-1;i++) for(int j=1;j<=N;j++) Tr[j][i]=Tr[Tr[j][i-1]]
    [i-1];//j节点向上2^i就是j向上2^i-1的节点在向上2^i-1
34    for(int i=1;i<=M;i++) X=read(), Y=read(), cout<<LCA(X,Y)<< '\n';
35    return 0;
36  }
```

Dinic 求最大流 (不知为什么这么快)

```
1   long long Ans(0), C;
2   int m, n, hd, tl, Dep[205];
3   struct Edge;
4   struct Node {
5     Edge *Fst[205], *Scd[205];
6     unsigned int Cntne;
7   } N[205], *S, *T, *A, *B, *Q[205];
8   struct Edge {
9     Node *To;
10    long long Mx, Nw;
11  } E[10005], *Cnte(E);
12  void Lnk(Node *x, Node *y, const long long &z) {
13    if (x->Fst[y - N]) {
14      x->Fst[y - N]->Mx += z;
15      return;
16    }
17    x->Fst[y - N] = Cnte;
18    Cnte->To = y;
19    Cnte->Mx = z;
20    (Cnte++)->Nw = 0;
21    return;
22  }
23  void BFS() {
24    Node *x;
25    while (hd < tl) {
26      x = Q[hd++];
27      if (x == T) continue;
28      for (register unsigned int i(1); i <= x->Cntne; i++) {
29        if (!Dep[x->Scd[i]->To - N] && x->Scd[i]->Nw < x->Scd[i]->Mx) {
30          Dep[x->Scd[i]->To - N] = Dep[x - N] + 1;
31          Q[tl++] = x->Scd[i]->To;
32        }
33      }
```

```cpp
    }
    return;
  }
  long long DFS(Node *x, long long Cm) {
    long long tmp, sum(0);
    for (register unsigned int i(1); i <= x->Cntne; i++) {
      if (x->Scd[i]->To == T) {  //汇点
        tmp = min(x->Scd[i]->Mx - x->Scd[i]->Nw, Cm);
        sum += tmp;
        x->Scd[i]->Nw += tmp;
        T->Fst[x - N]->Nw -= tmp;
        continue;
      }
      if (x->Scd[i]->Mx > x->Scd[i]->Nw &&
          Dep[x->Scd[i]->To - N] == Dep[x - N] + 1) {  //下一层的点
        if (Cm == 0) return sum;
        tmp = min(x->Scd[i]->Mx - x->Scd[i]->Nw, Cm);
        if (tmp = DFS(x->Scd[i]->To, tmp)) {
          Cm -= tmp;
          x->Scd[i]->Nw += tmp;
          x->Scd[i]->To->Fst[x - N]->Nw -= tmp;
          sum += tmp;
        }
      }
    }
    return sum;
  }
  void Dinic() {
    while (1) {
      memset(Q, 0, sizeof(Q)), memset(Dep, 0, sizeof(Dep));
      hd = 0, tl = 1, Q[hd] = S, Dep[S - N] = 1;
      BFS();
      if (!Dep[T - N]) break;
      Ans += DFS(S, 0x3f3f3f3f3f3f3f3f);
    }
    return;
  }
  int main() {
    n = RD(), m = RD(), S = RD() + N, T = RD() + N;
    memset(N, 0, sizeof(N));
    for (register unsigned int i(1); i <= m; ++i) {
      A = RD() + N, B = RD() + N, C = RD();
      Lnk(A, B, C), Lnk(B, A, 0);
    }
    for (register unsigned int i(1); i <= n; ++i) {
      N[i].Cntne = 0;
      for (register unsigned int j(1); j <= n; ++j) if (N[i].Fst[j]) N[i].Scd[++N[i].Cntne] = N[i].Fst[j];
    }
    Dinic(), printf("%llu\n", Ans);
    return 0;
  }
```

Kruskal

```
int n,m,fa[10005],s,e,l,k=0,ans=0;
struct side{int le,ri,len;}a[200005];
bool cmp(side x,side y){return(x.len<y.len);}
int find(int x){
  if(fa[x]==x) return x;
  fa[x]=find(fa[x]);
  return fa[x];
}
int main(){
  cin>>n>>m;
  memset(a,0x3f,sizeof(a));
  for(int i=1;i<=m;i++) cin>>s>>e>>l, a[i].le=s, a[i].ri=e, a[i].len=l;
  sort(a+1,a+m+1,cmp);
  for(int i=1;i<=n;i++) fa[i]=i;
  int i=0;
  while((k<n-1)&&(i<=m)){
    i++;
    int fa1=find(a[i].le),fa2=find(a[i].ri);
    if(fa1!=fa2) ans+=a[i].len, fa[fa1]=fa2, k++;
  }
  cout<<ans<<"\n";
  return 0;
}
```

Dijkstra

```
void Dijkstra() {
  q.push(make_pair(-N[s].Dst, s));
  Node *now;
  while (!q.empty()) {
    now = N + q.top().second;
    q.pop();
    if(now->InStk) continue;
    now->InStk = 1;//就是这里没判
    for (Edge *Sid(now->Fst); Sid; Sid = Sid->Nxt) {
      if (Sid->To->Dst < now->Dst + Sid->Val)
        if (!Sid->To->InStk) {//还有这
          Sid->To->Dst = now->Dst + Sid->Val;
          q.push(make_pair(Sid->To->Dst, Sid->To - N));
        }
    }
  }
  return;
}
```

Tarjan (强连通分量)

```
struct Edge;
struct Edge_;
struct Node N[10005], *Stk[10005];
struct Node_ N_[10005], *Stk_[10005];
struct Edge E[10005], *cnte(E);
struct Edge_ E_[10005], *cnte_(E_);
```

```cpp
int n, A, Dcnt(0), Dcnt_(0), Scnt(0), Hd(0), Hd_(0);
void Lnk_(const int &x, const int &y) {
  ++N_[y].IDg;
  (++cnte_)->To = N_ + y;
  cnte_->Nxt = N_[x].Fst;
  N_[x].Fst = cnte_;
  return;
}
void Tarjan(Node *x) {
  printf("To %d %d\n", x - N, Dcnt);
  if (!x->DFSr) {
    x->DFSr = ++Dcnt;
    x->BkT = x->DFSr;
    x->InStk = 1;
    Stk[++Hd] = x;
  }
  Edge *Sid(x->Fst);
  while (Sid) {
    if (Sid->To->BlT) {
      Sid = Sid->Nxt;
      continue;
    }
    if (Sid->To->InStk) x->BkT = min(x->BkT, Sid->To->DFSr);
    else {
      Tarjan(Sid->To);
      x-> BkT = min(x->BkT, Sid->To->BkT);
    }
    Sid = Sid->Nxt;
  }
  if(x->BkT == x->DFSr) {
    ++Scnt;
    while (Stk[Hd] != x) {
      Stk[Hd]->BlT = Scnt;
      Stk[Hd]->InStk = 0;
      --Hd;
    }
    Stk[Hd]->BlT = Scnt;
    Stk[Hd--]->InStk = 0;
  }
  return;
}
void ToPnt(Node *x) {
  Edge *Sid(x->Fst);
  while (Sid) {
    if(x->BlT != Sid->To->BlT) Lnk_(x->BlT, Sid->To->BlT);
    Sid = Sid->Nxt;
  }
  return;
}
```

拓扑序

```cpp
void TPR() {
  for(register int i(1); i <= Scnt; ++i) if (N_[i].IDg == 0) Stk_[++Hd_] = &N_[i];
  while (N_[++Hd_].IDg == 0) Stk_[Hd_] = &N_[Hd_];
  --Hd_;
```

```
 5    while (Hd_) {
 6      Stk_[Hd_]->Tpr = ++Dcnt_;
 7      Edge_ *Sid(Stk_[Hd_--]->Fst);
 8      while (Sid) {
 9        if(!Sid->To->Tpr) {
10          --(Sid->To->IDg);
11          if(Sid->To->IDg == 0) {
12            Stk_[++Hd_] = Sid->To;
13          }
14        }
15        Sid = Sid->Nxt;
16      }
17    }
18    return;
19  }
20  void DFS_(Node_ *x) {
21    x->_ed = 1;
22    printf("To %d %d\n", x - N_, x->Tpr);
23    Edge_ *Sid(x->Fst);
24    while (Sid) {
25      if(!Sid->To->_ed) DFS_(Sid->To);
26      Sid = Sid->Nxt;
27    }
28   return;
29  }
```

二分图最大匹配 (匈牙利)

```
 1  struct Edge;
 2  struct Node {
 3    bool Flg;
 4    Edge *Fst;
 5    Node *MPr;
 6  } L[505], R[505];
 7  struct Edge {
 8    Edge *Nxt;
 9    Node *To;
10  } E[50005], *cnte(E);
11  void Clr() { n = RD(), m = RD();
12    if (m < n) swap(m, n), flg = 1;
13    e = RD(), memset(L, 0, sizeof(L)), memset(R, 0, sizeof(R)), memset(E, 0,
    sizeof(E));
14    return; }
15  void Lnk(Node *x, Node *y) { (++cnte)->To = y, cnte->Nxt = x->Fst,  x->Fst =
    cnte;
16    return; }
17  bool Try(Node *x) {
18    x->Flg = 1;
19    Edge *Sid(x->Fst);
20    while (Sid) {
21      if (!(Sid->To->Flg)) {
22        if (Sid->To->MPr) {
23          if (Sid->To != x->MPr) {
24            if (!(Sid->To->MPr->Flg)) {
25              if (Try(Sid->To->MPr)) {
26                Sid->To->MPr = x;
27                x->MPr = Sid->To;
```

```
28            x->Flg = 0;
29            return 1;
30          } else Sid->To->Flg = 1;
31        }
32      }
33    } else {
34      Sid->To->MPr = x;
35      x->MPr = Sid->To;
36      ++ans;
37      return 1;
38    }
39   }
40   Sid = Sid->Nxt;
41  }
42  return 0;
43 }
44 int main() {
45  Clr();
46  for (register int i(1); i <= e; ++i) {
47    A = RD(), B = RD();
48    if (flg) swap(A, B);
49    Lnk(L + A, R + B);
50  }
51  for (register int i(1); i <= n; ++i) {
52    Edge *Sid(L[i].Fst);
53    while (Sid) {
54      if (Sid->To->MPr) {
55        if (Try(Sid->To->MPr)) {
56          Sid->To->MPr = L + i;
57          L[i].MPr = Sid->To;
58          Sid->To->Flg = 0;
59          break;
60        }
61      } else {
62        Sid->To->MPr = L + i;
63        L[i].MPr = Sid->To;
64        Sid->To->Flg = 0;
65        ++ans;
66        break;
67      }
68      Sid = Sid->Nxt;
69    }
70  }
71  printf("%d\n", ans);
72  return 0;
73 }
```

数学

Euclid (Gcd)

最基本的数学算法, 用来 $O(log_2 n)$ 求两个数的 GCD (最大公因数).

```
1  int Gcd(int x, int y) {
2    if(y == 0) return x;
3    return Gcd(y, x % y);
4  }
```

Exgcd

```
1  inline void Exgcd(int a, int b, int &x, int &y) {
2    if(!b) x = 1, y = 0;
3    else Exgcd(b, a % b, y, x), y -= (a / b) * x;
4  }
```

快速幂

```
1  unsigned Power(unsigned x, unsigned y) {
2    if(!y) return 1;
3    unsigned tmp(Power(x, y >> 1));
4    tmp = ((long long)tmp * tmp) % D;
5    if(y & 1) return ((long long)tmp * x) % D;
6    return tmp;
7  }
```

光速幂 (扩展欧拉定理)

```
1   unsigned Phi(unsigned x) {
2     unsigned tmp(x), anotherTmp(x), Sq(sqrt(x));
3     for (register unsigned i(2); i <= Sq && i <= x; ++i) {
4       if(!(x % i)) {
5         while (!(x % i)) x /= i;
6         tmp /= i, tmp *= i - 1;
7       }
8     }
9     if (x > 1) tmp /= x, tmp *= x - 1;//存在大于根号 x 的质因数
10    return tmp;
11  }
12  int main() {
13    A = RD(), D = RD(), C = Phi(D);
14    while (ch < '0' || ch > '9') ch = getchar();
15    while (ch >= '0' && ch <= '9') {
16      B *= 10, B += ch - '0';
17      if(B > C) flg = 1, B %= C;
18      ch = getchar();
19    }
20    if(B == 1) {
21      printf("%u\n", A % D);
22      return Wild_Donkey;
23    }
24    if(flg) printf("%u\n", Power(A, B + C));
25    else printf("%u\n", Power(A, B));
26    return Wild_Donkey;
27  }
```

矩阵快速幂

```cpp
struct Matrix {long long a[105][105], siz;}mtx;
long long k;bool flg;
Matrix operator*(Matrix x, Matrix y) {
  Matrix ans;
  long long tmp; ans.siz = x.siz;
  for (int i = 1; i <= ans.siz; i++) {
    for (int j = 1; j <= ans.siz; j++) {
      for (int k = 1; k <= ans.siz; k++) {
        tmp = x.a[k][j] * y.a[i][k];
        tmp %= 1000000007;
        ans.a[i][j] += tmp;
        ans.a[i][j] %= 1000000007;
      }
    }
  }
  return ans;
}
void print(Matrix x) {
  for (int i = 1; i <= x.siz; i++) {
    for (int j = 1; j <= x.siz; j++) printf("%lld ", x.a[i][j]);
    printf("\n");
  }
  return;
}
Matrix power(Matrix x, long long y) {
  Matrix ans, ans.siz = x.siz;
  if (y == 0) {
    for (int i = 1; i <= x.siz; i++) for (int j = 1; j <= x.siz; j++) if (i
== j) ans.a[i][j] = 1;
        else ans.a[i][j] = 0;
    return ans;
  }
  if (y == 1) return x;
  if (y == 2) return (x * x);
  if (y % 2) {  //奇次幂
    ans = power(x, y >> 1);
    return ans * ans * x;
  } else {
    ans = power(x, y >> 1);
    return ans * ans;
  }
  return ans;
}
int main() {
  scanf("%lld%lld", &mtx.siz, &k);
  for (int i = 1; i <= mtx.siz; i++) for (int j = 1; j <= mtx.siz; j++)
scanf("%lld", &mtx.a[i][j]);
  print(power(mtx, k));
  return 0;
}
```

线性求逆元

```
1  signed main() {
2    n = RD(), p = RD(), a[1] = 1, write(a[1]);
3    for (register unsigned i(2); i <= n; ++i) a[i] = ((long long)a[p % i] * (p
     - p / i)) % p, write(a[i]);
4    fwrite(_d,1,_p-_d,stdout);
5    return Wild_Donkey;
6  }
```

欧拉筛 (线性筛)

```
1  vis[1]=1;
2  for(int i=2;i<=n;i++)//枚举倍数
3  {
4    if(!vis[i]) prime[cnt++]=i;//i无最小因子 ，i就是下一个质数(从0开始)
5    for(int j=0;j<cnt&&i*prime[j]<=n;j++)//（枚举质数）保证prime访问到的元素是已经筛
     出的质数
6    {
7      vis[i*prime[j]]=prime[j]; //第j个质数的i倍数不是质数
8      if(i%prime[j]==0) break;
9    }
10 }
```

P.S. 可以用来线性求积性函数

Lucas_Law (C(n, n + m) % p)

```
1  unsigned a[10005], m, n, Cnt(0), A, B, C, D, t, Ans(0), Tmp(0), Mod;
2  bool b[10005];
3  unsigned Power (unsigned x, unsigned y) {
4    if(!y) {
5      return 1;
6    }
7    unsigned tmp(Power(((long long)x * x) % Mod, y >> 1));
8    if(y & 1) return ((long long)x * tmp) % Mod;
9    return tmp;
10 }
11 unsigned Binom (unsigned x, unsigned y) {
12   unsigned Up(1), Down(1);
13   if (y > x) return 0;
14   if(!y) return 1;
15   for (register unsigned i(2); i <= x; ++i) Up = ((long long)Up * i) % Mod;
16   for (register unsigned i(2); i <= y; ++i) Down = ((long long)Down * i) %
     Mod;
17   for (register unsigned i(2); i <= x - y; ++i) Down = ((long long)Down * i)
     % Mod;
18   Down = Power(Down, Mod - 2);
19   return ((long long)Up * Down) % Mod;
20 }
21 unsigned Lucas (unsigned x, unsigned y) {
22   if(y > x) return 0;
23   if(x <= Mod && y <= Mod) return Binom(x, y);
24   return ((long long)Binom(x % Mod, y % Mod) * Lucas(x / Mod, y / Mod)) %
     Mod;
```

```
25 }
26 int main() {
27   t = RD();
28   for (register unsigned T(1); T <= t; ++T){
29     n = RD(), m = RD(), Mod = RD();
30     if(!(n && m)) {
31       printf("1\n");
32       continue;
33     }
34     printf("%u\n", Lucas(n + m, n));
35   }
36   return Wild_Donkey;
37 }
```

字符串

KMP

```
1  int main() {
2    inch = getchar();
3    while (inch < 'A' || inch > 'Z') inch = getchar();
4    while (inch >= 'A' && inch <= 'Z') A[++la] = inch, inch = getchar();
5    while (inch < 'A' || inch > 'Z') inch = getchar();
6    while (inch >= 'A' && inch <= 'Z') B[++lb] = inch, inch = getchar();
7    unsigned k(1);
8    for (register unsigned i(2); i <= lb; ++i)  { // Origin_Len
9      while ((B[k] != B[i] && k > 1) || k > i) k = a[k - 1] + 1;
10     if(B[k] == B[i]) a[i] = k, ++k;
11     continue;
12   }
13   k = 1;
14   for (register unsigned i(1); i + lb <= la + 1;) {  // Origin_Address
15     while (A[i + k - 1] == B[k] && k <= lb) ++k;
16     if(k == lb + 1) printf("%u\n", i);
17     if(a[k - 1] == 0) {
18       ++i, k = 1;
19       continue;
20     }
21     --k, i += k - a[k], k = a[k] + 1;  // Substring of Len(k - 1) has
   already paired, so the next time, start with the border of the (k - 1)
   length substring
22   }
23   for (register unsigned i(1); i <= lb; ++i) printf("%u ", a[i]); //
   Origin_Len
24   return 0;
25 }
```

ACM (二次加强)

```
1  unsigned n, L(0), R(0), Tmp(0), Cnt(0);
2  char inch;
3  struct Node;
4  struct Edge {
5    Edge *Nxt;
6    Node *To;
```

```
 7  }E[200005], *Cnte(E);
 8  struct Node {
 9    Node *Son[26], *Fa, *Fail;
10    char Ch;
11    Edge *Fst;
12    bool Exist;
13    unsigned Size, Times;
14  }N[200005], *Q[200005], *now(N), *Cntn(N), *Find(N), *Ans[200005];
15  unsigned DFS(Node *x) {
16    Edge *Sid(x->Fst);
17    x->Size = x->Times;
18    now = x;
19    while (Sid) {
20      now = Sid->To;
21      x->Size += DFS(now);
22      Sid = Sid->Nxt;
23    }
24    return x->Size;
25  }
26  int main() {
27    n = RD();
28    for (register unsigned i(1); i <= n; ++i) {
29      while (inch < 'a' || inch > 'z') inch = getchar();//跳过无关字符
30      now = N;  // 从根开始
31      while (inch >= 'a' && inch <= 'z') {
32        inch -= 'a';     // 字符转化为下标
33        if(!(now->Son[inch])) now->Son[inch] = ++Cntn, Cntn->Ch = inch, Cntn-
    >Fa = now; // 新节点
34        now = now->Son[inch], inch = getchar(); // 往下走
35      }
36      if (!(now->Exist)) now->Exist = 1; //新串（原来不存在以这个点结尾的模式串）
37      Ans[i] = now; // 记录第 i 个串尾所在节点
38    }
39    for (register short i(0); i < 26; ++i) {   //  对第一层的特殊节点进行边界处理
40      if(N->Son[i]) {               // 根的儿子
41        Q[++R] = N->Son[i];        // 入队
42        N->Son[i]->Fail = N;     // Fail 往上连，所以只能连向根
43        (++Cnte)->Nxt = N->Fst; // 反向边，用边表存
44        N->Fst = Cnte;
45        Cnte->To = N->Son[i];
46      }
47    }
48    while (L < R) { // BFS 连边，建自动机
49      now = Q[++L]; // 取队首并弹出
50      for (register short i(0); i < 26; ++i) if(now->Son[i]) Q[++R] = now-
    >Son[i];
51      if(!(now->Fa)) continue;
52      Find = now->Fa->Fail; // 从父亲的 Fail 开始往上跳，直到找到
53      while (Find) {
54        if(Find->Son[now->Ch]) {                  // 找到了（边界）
55          now->Fail = Find->Son[now->Ch];         // 正向边（往上连）
56          (++Cnte)->Nxt = Find->Son[now->Ch]->Fst;  // 反向边（往下连）
57          Find->Son[now->Ch]->Fst = Cnte;
58          Cnte->To = now;
59          break;
60        }
61        Find = Find->Fail;  // 继续往前跳
62      }
```

```
63      if(!(now->Fail)) {
64        now->Fail = N;  // 所有找不到对应 Fail 的节点，Fail 均指向根
65        (++Cnte)->Nxt = N->Fst;
66        N->Fst = Cnte;
67        Cnte->To = now;
68      }
69    }
70    while (inch < 'a' || inch > 'z') {
71      inch = getchar();
72    }
73    now = N;
74    while (inch >= 'a' && inch <= 'z') {   // 自动机扫一遍
75      inch -= 'a';
76      if(!now) now = N; // 如果完全失配了，则从根开始新的匹配，否则接着前面已经匹配成功
   的节点继续匹配
77        while(now) {                    // 完全失配，跳出
78          if(now->Son[inch]) {      // 匹配成功，同样跳出
79            now = now->Son[inch]; // 自动机对应节点和字符串同步往下走
80            ++(now->Times);          // 记录节点扫描次数
81            break;
82          }
83          now = now->Fail;          // 跳 Fail
84        }
85      inch = getchar();
86    }
87    DFS(N); // 统计互相包含的模式串
88    for (register unsigned i(1); i <= n; ++i) printf("%u\n", Ans[i]->Size); //
   根据之前记录的第 i 个模式串尾字符对应的节点的指针找到需要的答案
89    return 0;
90 }
```

SA

```
1  unsigned m, n, Cnt(0), A, B, C, D, t, Ans(0), Tmp(0), Bucket[1000005],
   sumBucket[1000005], Tmpch[64], a[1000005], b[1000005];
2  char Inch[1000005];
3  struct Suffix {
4    unsigned RK, SubRK;
5  }S[1000005], Stmp[1000005];
6  void RadixSort () {
7    unsigned MX(0);                                          // 记录最大键值
8    for (register unsigned i(1); i <= n; ++i) {
9      ++Bucket[S[i].SubRK];                                  // 第二关键字入桶
10     MX = max(S[i].SubRK, MX);
11   }
12   sumBucket[0] = 0;
13   for (register unsigned i(1); i <= MX; ++i) {            // 求前缀和以确定在排序后
   的序列中的位置
14     sumBucket[i] = sumBucket[i - 1] + Bucket[i - 1];   // 求桶前缀和，前缀和右边
   界是开区间，所以计算的是比这个键值小的所有元素个数
15     Bucket[i - 1] = 0;                                    // 清空桶
16   }
17   Bucket[MX] = 0;
18   for (register unsigned i(1); i <= n; ++i) {             // 排好的下标存到 b 中，
   即 b[i] 为第 i 小的后缀编号
19     b[++sumBucket[S[i].SubRK]] = i;                       // 前缀和自增是因为
20   }
```

```cpp
      b[0] = 0;                                              // 边界（第 0 小的不存在）
      for (register unsigned i(1); i <= n; ++i) {
        a[i] = b[i];
      }
      MX = 0;
      for (register unsigned i(1); i <= n; ++i) {
        ++Bucket[S[i].RK];                                   // 第一关键字入桶
        MX = max(S[i].RK, MX);
      }
      sumBucket[0] = 0;
      for (register unsigned i(1); i <= MX; ++i) {
        sumBucket[i] = sumBucket[i - 1] + Bucket[i - 1];
        Bucket[i - 1] = 0;
      }
      Bucket[MX] = 0;
      for (register unsigned i(1); i <= n; ++i) {
        b[++sumBucket[S[a[i]].RK]] = a[i];                   // 由于 a[i] 是 b[i] 的
  拷贝，表示第 i 小的后缀编号，所以枚举 i 一定是从最小的后缀开始填入新意义下的 b
      }
      b[0] = 0, Cnt = 0;// 使 RK 不那么分散
      for (register unsigned i(1); i <= n; ++i) {
        if(S[b[i]].SubRK != S[b[i - 1]].SubRK || S[b[i]].RK != S[b[i - 1]].RK)
  a[b[i]] = ++Cnt;// 第 i 小的后缀和第 i - 1 小的后缀不等排名不等
        else a[b[i]] = Cnt;// 第 i 小的后缀和第 i - 1 小的后缀相等排名也相等
      }
      for (register unsigned i(1); i <= n; ++i) S[i].RK = a[i]// 将 a 中暂存的新次
  序拷贝回来
      return;
  }
  int main() {
    cin.getline(Inch, 1000001);
    n = strlen(Inch);
    for (register unsigned i(0); i < n; ++i) {
      if(Inch[i] <= '9' && Inch[i] >= '0') {Inch[i] -= 47;continue;}
      if(Inch[i] <= 'Z' && Inch[i] >= 'A') {Inch[i] -= 53;continue;}
      if(Inch[i] <= 'z' && Inch[i] >= 'a') {Inch[i] -= 59;continue;}
    }
    for (register unsigned i(0); i < n; ++i) Bucket[Inch[i]] = 1;
    for (register unsigned i(0); i < 64; ++i) {
      if(Bucket[i]) {
        Tmpch[i] = ++Cnt; // 让桶从 1 开始，空出 0 的位置
        Bucket[i] = 0;
      }
    }
    for (register unsigned i(0); i < n; ++i) S[i + 1].RK = Tmpch[Inch[i]];//
  将字符串离散化成整数序列，字符串读入是 [0, n) 的，题意中字符串是 (0, n] 的
    for (register unsigned i(1); i <= n; i <<= 1) {        // 当前按前 i 个字符排完
  了，每次 i 倍增
      for (register unsigned j(1); j + i <= n; ++j) S[j].SubRK = S[j +
  i].RK;// 针对第二关键字不为 0 的
      for (register unsigned j(n - i + 1); j <= n; ++j) S[j].SubRK = 0;// 第二
  关键字为 0
      RadixSort();
    }
    for (register unsigned i(1); i <= n; ++i) b[S[i].RK] = i;
    for (register unsigned i(1); i <= n; ++i) printf("%u ", b[i]);
    return Wild_Donkey;
  }
```

SA-IS

```cpp
unsigned Cnt(0), n, Ans(0), Tmp(0), SPool[2000005], SAPool[2000005],
BucketPool[2000005], SumBucketPool[2000005], AddressPool[2000005],
S_S1Pool[2000005];
char TypePool[2000005];
inline char Equal (unsigned *S, char *Type, unsigned x, unsigned y) {
  while (Type[x] & Type[y]) {     // 比较 S 区
    if(S[x] ^ S[y]) return 0;
    ++x,++y;
  }
  if(Type[x] | Type[y]) return 0; // L 区起点是否整齐
  while (!(Type[x] | Type[y])) {   // 比较 L 区
    if(S[x] ^ S[y]) return 0;
    ++x, ++y;
  }
  if(Type[x] ^ Type[y]) return 0; // 尾 S 位置是否对应
  if(S[x] ^ S[y]) return 0;       // 尾 S 权值是否相等
  return 1;
}
void Induc (unsigned *Address, char *Type, unsigned *SA, unsigned *S,
unsigned *S_S1, unsigned *Bucket, unsigned *SumBucket, unsigned N);// 诱导
SA
void Induced_Sort (unsigned *Address, char *Type, unsigned *SA, unsigned
*S, unsigned *S_S1, unsigned *Bucket, unsigned *SumBucket, unsigned N,
unsigned bucketSize, unsigned LMSR) {// 通过 S 求 SA
  SumBucket[0] = 1;
  for (register unsigned i(1); i <= bucketSize; ++i)    // 重置每个栈的栈底
(右端)
    SumBucket[i] = SumBucket[i - 1] + Bucket[i];
  memset(SA + 1, 0, sizeof(unsigned) * N);              // 在上一层的诱导排序
中，填入了 SA，这里进行清空
  for (register unsigned i(LMSR); i > N; --i)           // 放长度为 1 的 LMS
前缀
    SA[SumBucket[S[Address[i]]]--] = Address[i];
  SumBucket[0] = 1;
  for (register unsigned i(1); i <= bucketSize; ++i)    // 重置每个栈的栈底
(左端)
    SumBucket[i] = SumBucket[i - 1] + Bucket[i];
  for (register unsigned i(1); i <= N; ++i)             // 从左到右扫 SA 数组
    if(SA[i] && (SA[i] - 1))
      if(!Type[SA[i] - 1])                              // Suff[SA[i] - 1]
是 L-Type
        SA[++SumBucket[S[SA[i] - 1] - 1]] = SA[i] - 1;
  SumBucket[0] = 1;
  for (register unsigned i(1); i <= bucketSize; ++i)    // 重置每个栈的栈底
(右端)
    SumBucket[i] = SumBucket[i - 1] + Bucket[i];
  for (register unsigned i(N); i >= 1; --i)             // 从右往左扫 SA 数组
    if(SA[i] && (SA[i] - 1))
      if(Type[SA[i] - 1])                               // Suff[SA[i] - 1]
是 S-Type
        SA[SumBucket[S[SA[i] - 1]]--] = SA[i] - 1;
  register char flg(0)/*是否有重*/;
  register unsigned CntLMS(0)/*本质不同的 LMS 子串数量*/, Pre(N)/*上一个 LMS 子
串起点*/, *Pointer(SA + N + 1)/*LMS 子串的 SA 的头指针*/;
```

```
41    for (register unsigned i(2); i <= N; ++i)              // 扫描找出 LMS，判重
并命名
42      if(Type[SA[i]] && (!Type[SA[i] - 1])) {
43        if(Pre ^ N && Equal(S, Type, SA[i], Pre))          // 暴力判重
44          S[S_S1[SA[i]]] = CntLMS, flg = 1;                // 命名
45        else S[S_S1[SA[i]]] = ++CntLMS;                    // 命名
46        Pre = SA[i];                                       // 用来判重
47        *(++Pointer) = S_S1[SA[i]] - N;                    // 记录 LMS
48      }
49    S[LMSR] = 0, SA[N + 1] = LMSR - N;                     // 末尾空串最小
50    if(flg)                                                // 有重复 LMS 子串，
递归排序 S1
51      Induc(Address + N, Type + N, SA + N, S + N, S_S1 + N, Bucket +
bucketSize + 1, SumBucket + bucketSize + 1, LMSR - N); //有重复，先诱导 SA1，
新的 Bucket 直接接在后面
52    return;                                                // 递归跳出，保证 SA1
是严格的
53  }
54  void Induc (unsigned *Address, char *Type, unsigned *SA, unsigned *S,
unsigned *S_S1, unsigned *Bucket, unsigned *SumBucket, unsigned N) {// 诱导
SA
55    for (register unsigned i(1), j(1); i < N; ++i) {       // 定性 S/L
56      if(S[i] < S[i + 1]) while (j <= i) Type[j++] = 1;    // Suff[j~i] 是 S-
Type
57      if(S[i] > S[i + 1])                                  // Suff[j~i] 是 L-
Type
58        while (j <= i) Type[j++] = 0;
59    }
60    Type[N] = 1, Type[0] = 1;
61    register unsigned CntLMS(N)/*记录 LMS 字符数量*/;
62    for (register unsigned i(1); i < N; ++i)               // 记录 S1 中字符对应
的 S 的 LMS 子串左端 LMS 字符的位置 Address[]，和 S 中的 LMS 子串在 S1 中的位置
S_S1[]
63      if(!Type[i]) if(Type[i + 1])
64        Address[++CntLMS] = i + 1, S_S1[i + 1] = CntLMS;
65    register unsigned bucketSize(0);                       // 本次递归字符集大小
66    for (register unsigned i(1); i <= N; ++i)              // 确定 Bucket，可以
线性生成 SumBucket
67      ++Bucket[S[i]], bucketSize = bucketSize < S[i] ? S[i] : bucketSize; //
统计 Bucket 的空间范围
68    Induced_Sort(Address, Type, SA, S, S_S1, Bucket, SumBucket, N,
bucketSize, CntLMS);// 诱导排序 LMS 子串，求 SA1
69    memset(SA + 1, 0, sizeof(unsigned) * N);               // 在求 SA1 时也填了
一遍 SA，这里进行清空
70    SumBucket[0] = 1;                                      // SA1 求出来了，开始
诱导 SA
71    for (register unsigned i(1); i <= bucketSize; ++i)     // 重置每个栈的栈底
(右端)
72      SumBucket[i] = SumBucket[i - 1] + Bucket[i];
73    for (register unsigned i(CntLMS); i > N; --i)          // 放 LMS 后缀
74      SA[SumBucket[S[Address[SA[i] + N]]]--] = Address[SA[i] + N];
75    SumBucket[0] = 1;
76    for (register unsigned i(1); i <= bucketSize; ++i)     // 重置每个栈的栈底
(左端)
77      SumBucket[i] = SumBucket[i - 1] + Bucket[i];
78    for (register unsigned i(1); i <= N; ++i)  // 从左到右扫 SA 数组
79      if(SA[i] && (SA[i] - 1))
```

```cpp
      if(!Type[SA[i] - 1]) SA[++SumBucket[S[SA[i] - 1] - 1]] = SA[i] - 1;
// Suff[SA[i] - 1] 是 L-Type
    SumBucket[0] = 1;
    for (register unsigned i(1); i <= bucketSize; ++i) SumBucket[i] =
SumBucket[i - 1] + Bucket[i];// 重置每个栈的栈底（右端）
    for (register unsigned i(N); i >= 1; --i)// 从右往左扫 SA 数组
      if(SA[i] && (SA[i] - 1)) if(Type[SA[i] - 1])
        SA[SumBucket[S[SA[i] - 1]]--] = SA[i] - 1; // Suff[SA[i] - 1] 是 S-
Type
    return;
}
int main() {
    fread(TypePool + 1, 1, 1000004, stdin);
    for (register unsigned i(1); ; ++i) {    // 尽量压缩字符集
      if(TypePool[i] <= '9' && TypePool[i] >= '0') {SPool[i] = TypePool[i] -
47;continue;}
      if(TypePool[i] <= 'Z' && TypePool[i] >= 'A') {SPool[i] = TypePool[i] -
53;continue;}
      if(TypePool[i] <= 'z' && TypePool[i] >= 'a') {
        SPool[i] = TypePool[i] - 59;
        continue;
      }
      n = i;
      break;
    }
    SPool[n] = 0;// 最后一位存空串，作为哨兵
    Induc (AddressPool, TypePool, SAPool, SPool, S_S1Pool, BucketPool,
SumBucketPool, n);
    for (register unsigned i(2); i <= n; ++i) printf("%u ", SAPool[i]);//
SA[1] 是最小的后缀，算法中将空串作为最小的后缀，所以不输出 SA[1]
    return Wild_Donkey;
}
```

SAM

```cpp
unsigned m, n, Cnt(0), t, Ans(0), Tmp(0);
short nowCharacter;
char s[1000005];
struct Node {
  unsigned Length, Times; // 长度(等价类中最长的)，出现次数
  char endNode;           // 标记 (char 比 bool 快)
  Node *backToSuffix, *SAMEdge[26];
}N[2000005], *CntN(N), *Last(N), *now(N), *A, *C_c;
inline unsigned DFS(Node *x) {
  unsigned tmp(0);
  if(x->endNode) {
    tmp = 1;
  }
  for (register unsigned i(0); i < 26; ++i)
    if(x->SAMEdge[i])                    // 有转移 i
      if(x->SAMEdge[i]->Times > 0) tmp += x->SAMEdge[i]->Times;// 被搜索过，直
接统计
      else tmp += DFS(x->SAMEdge[i]); // 未曾搜索，搜索
  if (tmp > 1) Ans = max(Ans, tmp * x->Length);// 出现次数不为 1，尝试更新答案
  x->Times = tmp;                      // 存储子树和
  return tmp;                          // 子树和用于搜索树上的父亲的统计
}
```

```
22   int main() {
23     scanf("%s", s);
24     n = strlen(s);
25     for (register unsigned i(0); i < n; ++i) {
26       Last = now;                                    // Last 指针往后移
27       A = Last;                                      // s 对应的节点
28       nowCharacter = s[i] - 'a';                     // 取字符, 转成整数
29       now = (++CntN);                                // s + c 对应的节点
30       now->Length = Last->Length + 1;                // len[s + c] = len[s]
31       while (A && !(A->SAMEdge[nowCharacter])) {     // 跳到 A 有转移 c 的祖先
32         A->SAMEdge[nowCharacter] = now;              // 没有转移 c, 创造转移
     (Endpos = {len_s + 1})
33         A = A->backToSuffix;
34       }
35       if(!A) {                                       // c 首次出现
36         now->backToSuffix = N;                       // 后缀链接连根
37         continue;                                    // 直接进入下一个字符的加
     入
38       }
39       if (A->Length + 1 == A->SAMEdge[nowCharacter]->Length) {
40         now->backToSuffix = A->SAMEdge[nowCharacter];   // len[a] + 1 = len[a-
     >c] 无需分裂
41         continue;
42       }
43       (++CntN)->Length = A->Length + 1;              // 分裂出一个新点
44       C_c = A->SAMEdge[nowCharacter];                // 原来的 A->c 变成 C->c
45       memcpy(CntN->SAMEdge, C_c->SAMEdge, sizeof(CntN->SAMEdge));
46       CntN->backToSuffix = C_c->backToSuffix;        // 继承转移, 后缀链接
47       C_c->backToSuffix = CntN;                      // C -> c 是 A -> c 后
     缀链接树上的儿子
48       now->backToSuffix = CntN;                      // 连上 s + c 的后缀链接
49       while (A && A->SAMEdge[nowCharacter] == C_c) { // 这里要将 A 本来转移到
     C->c 的祖先重定向到 A->c
50         A->SAMEdge[nowCharacter] = CntN;             // 连边
51         A = A->backToSuffix;                         // 继续往上跳
52       }
53     }
54     while (now != N) {                               // 打标记
55       now->endNode = 1;                              // 从 s 向上跳 (从 s 到
     root 这条链上都是结束点)
56       now = now->backToSuffix;
57     }
58     DFS(N);                                          // 跑 DFS, 统计 + 更新
     Ans
59     printf("%u\n", Ans);
60     return Wild_Donkey;
61   }
```

GSAM (新的构造方式, 原理不变)

```
1    int main() {
2      n = RD(), N[0].Length = 0;
3      for (register unsigned i(1); i <= n; ++i) {     // 读入 + 建 Trie
4        scanf("%s", s);                               // 字符转成自然数
5        len = strlen(s);
6        now = N;
7        for (register unsigned j(0); j < len; ++j) {
```

```cpp
        s[j] -= 'a';
        if(!(now->To[s[j]])) {
          now->To[s[j]] = ++CntN;
          CntN->Father = now;
          CntN->Character = s[j];
          CntN->Length = now->Length + 1;              // 顺带着初始化一些信息
        }
        now = now->To[s[j]];
      }
    }
    Queue[++queueTail] = N;                             // 初始化队列，准备 BFS
    while (queueHead < queueTail) {                     // 简单的 BFS
      now = Queue[++queueHead];
      for (register char i(0); i < 26; ++i)
        if(now->To[i])
          if(!(now->To[i]->Visited))
            Queue[++queueTail] = now->To[i], now->To[i]->Visited = 1;
    }
    for (register unsigned i(2); i <= queueTail; ++i) { // BFS 留下的队列便是
BFS 序，这便是一个普通的后缀自动机构建
      now = Queue[i];                                   // 按队列的顺序进行插入，
保证 Link 跳到的节点已经插入
      A = now->Father;
      while (A && !(A->toAgain[now->Character])) {      // 跳 Link 边 + 连转移边
        A->toAgain[now->Character] = 1;                 // 原来的 Trie 边不代表
GSAM 边，这里的 toAgain 为真才说明 GSAM 有这个转移
        A->To[now->Character] = now;
        A = A->Link;
      }
      if(!A){now->Link = N;continue;} // 无对应字符转移
      if((A->Length + 1) ^ (A->To[now->Character]->Length)) {
        C_c = A->To[now->Character];
        (++CntN)->Length = A->Length + 1;               // 调了好久的问题，不要在
重定向之前自作主张提前转移 A->C
        CntN->Link = C_c->Link;
        memcpy(CntN->To, C_c->To, sizeof(C_c->To));
        memcpy(CntN->toAgain, C_c->toAgain, sizeof(C_c->toAgain));
        now->Link = CntN, C_c->Link = CntN;
        CntN->Character = C_c->Character;
        while (A && A->To[C_c->Character] == C_c) A->To[C_c->Character] =
CntN, A = A->Link;
        continue;
      }
      now->Link = A->To[now->Character];                // 连续转移，直接连 Link
    }
    for (register Node *i(N + 1); i <= CntN; ++i) Ans += i->Length - i->Link->Length;// 统计字串数
    printf("%llu\n", Ans);
    return Wild_Donkey;
}
```

Manacher

```
unsigned n, Frontier(0), Ans(0), Tmp(0), f1[11000005], f2[11000005];
char a[11000005];
int main() {
  fread(a+1,1,11000000,stdin);        // fread 优化
  n = strlen(a + 1);                  // 字符串长度
  a[0] = 'A';
  a[n + 1] = 'B';                     // 哨兵
  for (register unsigned i(1); i <= n; ++i) {    // 先求 f1
    if(i + 1 > Frontier + f1[Frontier]) {        // 朴素
      while (!(a[i - f1[i]] ^ a[i + f1[i]])) ++f1[i];
      Frontier = i;                              // 更新 Frontier
    }
    else {
      register unsigned Reverse((Frontier << 1) - i), A(Reverse -
f1[Reverse]), B(Frontier - f1[Frontier]);
      f1[i] = Reverse - ((A < B) ? B : A);                // 确定 f1[i]
下界
      if (!(Reverse - f1[Reverse] ^ Frontier - f1[Frontier])) { // 特殊情况
        while (!(a[i - f1[i]] ^ a[i + f1[i]])) ++f1[i];
        Frontier = i;                                     // 更新
Frontier
      }
    }
    Ans = ((Ans < f1[i]) ? f1[i] : Ans);
  }
  Ans = (Ans << 1) - 1;                          // 根据 max(f1) 求长度
  Frontier = 0;
  for (register unsigned i(1); i <= n; ++i) {
    if(i + 1 > Frontier + f2[Frontier]) {        // 朴素
      while (!(a[i - f2[i] - 1] ^ a[i + f2[i]])) ++f2[i];
      Frontier = i;                              // 更新 Frontier
    }
    else {
      register unsigned Reverse ((Frontier << 1) - i - 1), A(Reverse -
f2[Reverse]), B(Frontier - f2[Frontier]);
      f2[i] = Reverse - ((A < B) ? B : A);       // 确定 f2[i] 下界
      if (A == B) { // 特殊情况，朴素
        while (a[i - f2[i] - 1] == a[i + f2[i]]) {
          ++f2[i];
        }
        Frontier = i;                            // 更新 Frontier
      }
    }
    Tmp = ((Tmp < f2[i]) ? f2[i] : Tmp);
  }
  Tmp <<= 1;                                     // 根据 max(f2) 求长度
  printf("%u\n", (Ans < Tmp) ? Tmp : Ans);       // 奇偶取其大
  return Wild_Donkey;
}
```

PAM

```cpp
unsigned m, n, Cnt(0), Ans(0), Tmp(0), Key;
bool flg(0);
char a[500005];
struct Node {
  Node *Link, *To[26];
  int Len;
  unsigned int LinkLength;
}N[500005], *Order[500005], *CntN(N + 1), *Now(N), *Last(N);
int main() {
  fread(a + 1, 1, 500003, stdin);
  n = strlen(a + 1);
  N[0].Len = -1;
  N[1].Link = N;
  N[1].Len = 0;
  Order[0] = N + 1;
  for (register unsigned i(1); i <= n; ++i) {
    if(a[i] < 'a' || a[i] > 'z') continue;
    Now = Last = Order[i - 1];
    a[i] -= 'a';
    a[i] = ((unsigned)a[i] + Key) % 26;
    while (Now) {
      if(Now->Len + 1 < i) {
        if(a[i - Now->Len - 1] == a[i]) {
          if(Now->To[a[i]]) {
            Order[i] = Now->To[a[i]];
            flg = 1;
          }
          else {
            Now->To[a[i]] = ++CntN;
            CntN->Len = Now->Len + 2;
            Order[i] = CntN;
          }
          break;
        }
      }
      Last = Now, Now = Now->Link;
    }
    if(!flg) {
      Now = Last;
      while (Now) {
        if(Now->To[a[i]]) {
          if(Now->To[a[i]]->Len < Order[i]->Len) {
            if(a[i - Now->Len - 1] == a[i]) {
              Order[i]->Link = Now->To[a[i]];
              Order[i]->LinkLength = Now->To[a[i]]->LinkLength + 1;
              break;
            }
          }
        }
        Now = Now->Link;
      }
      if(!Now) {
        Order[i]->Link = N + 1;
        Order[i]->LinkLength = 1;
      }
```

```
56        }
57      else flg = 0;
58      Key = Order[i]->LinkLength;
59      printf("%d ", Key);
60    }
61    return Wild_Donkey;
62  }
```

Stl 或库函数的用法

一些 Stl 需要传入数组里操作位置的头尾指针, 遵循左闭右开的规则, 如 `(A + 1, A + 3)` 表示的是 $A[1]$ 到 $A[2]$ 范围. 在包含了前面提到的头文件后, 可以正常使用.

sort

`sort(A + 1, A + n + 1)`

表示将 A 数组从 $A[1]$ 到 $A[n]$ 升序排序.

其他规则可通过重载结构体的 $<$ 或比较函数 `Cmp()` 来定义.

下面放一个归并排序 (Merge) 的板子

```
1  int n,a[100005],b[100005];
2  void merge(int l,int m,int r) {  //1~m是有序的,m~r是有序的
3    int i=l,j=m+1;
4    for(int k=1;k<=r-l+1;++k)
5      if(i>m) b[k]=a[j++];
6      else  if(j>r) b[k]=a[i++];
7            else  if(a[i]<a[j]) b[k]=a[i++];
8                  else b[k]=a[j++];
9    for(int k=1;k<=r-l+1;++k) a[l+k-1]=b[k];
10  }
11  void mergesort(int l,int r) {
12    if(l==r) return;
13    int m=(l+r)/2;
14    mergesort(l,m), mergesort(m+1,r), merge(l,m,r);
15  }
16  int main() {
17    scanf("%d",&n);
18    for(int i=1;i<=n;++i) scanf("%d",a+i);
19    mergesort(1,n); //归并排序[1,n]
20    for(int i=1;i<=n;++i) printf("%d%c",a[i],i==n?'\n':' ');
21    return 0;
22  }
```

priority_queue

`priority_queue <int> q`

定义一个元素为 `int` 的默认优先队列 (二叉堆)

`q.push(x)`

$O(log_2 n)$ 插入元素 x

`q.pop()`

$O(log_2 n)$ 弹出堆顶

`q.top()`

$O(1)$ 查找堆顶, 返回值为队列元素类型

默认容器为对应数据类型的 `<vector>`, 一般不需要修改, 也可以通过重载 $<$ 或比较函数来定义规则

lower/upper_bound

`lower_bound(A + 1, A + n + 1, x)` 查找有序数组 A 中从 $A[1]$ 到 $A[n]$ 范围内最小的大于等于 x 的数的迭代器.

`upper_bound()` 同理, 只是大于等于变成了严格大于, 其他用法都相同

值得一提的是, 如果整个左闭右开区间内都没有找到合法的元素, 那么返回值将会是传入区间的右端点, 而右端点又恰恰不会被查询区间包含, 这样如果找不到, 它的返回值将不会和任意一个合法结果产生歧义.

也可以用一般方法定义比较规则.

set

`set <Type> A` 定义, 需定义 `Type` 的小于号, 用平衡树 (RBT) 维护集合, 支持如下操作:

- 插入元素

`A.insert(x)` 插入一个元素 x

- 删除元素

`A.erase(x)` 删除存在的元素 x

- 查询元素

`A.find(x)` 查询是否存在, 存在返回 `true`

- 头尾指针 (迭代器)

`A.begin()` `A.end()`, 返回整个集合的最值

另外, 也支持 `A.lower/upper_bound()` 查询相邻元素.

multiset

在 set 的基础上支持重复元素 (违背了集合的数学定义, 但是能解决特定问题)

- 元素计数

`A.count(x)` 返回对应元素的个数.

pair

使用 `make_pair(x, y)` 代表一个对应类型的组合.

`pair<Type1, Type2> A` 定义组合 A. $A.first, A.second$ 分别是类型为 `Type1`, `Type2` 的两个变量.

map

`map <Type1, Type2> A` 定义一个映射, 用前者类型的变量作为索引, 可以 $O(logn)$ 检索后者变量的地址. 要定义 `Type1` 的小于号.

用法类似于 `set`, 但是操作的键值是 $Type1$ 类型的, 访问到的是 $pair < Type1, Type2 >$ 类型的迭代器.

unordered_map

基于哈希的映射而不是平衡树, 好处是 $O(1)$ 操作, 坏处是键值没有大小关系的区分, 也就是说 `set` 作为平衡树的功能 (前驱/后继, 最值, 迭代器自增/减)

memset

`memset(A, 0x00, x * sizeof(Type))` 将 A 中前 x 个元素的每个字节都设置为 `0x00`

这里可以用 `sizeof(A)` 对整个数组进行设置.

memcpy

`memcpy(A, B, x * sizeof(Type))` 将 B 的前 x 个元素复制到 A 的对应位置. 代替 `for` 循环, 减小常数.

fread

`fread(A, 1, x, stdin)`, 将 x 个字符读入 A 中, 一般用来读入字符串. 速度极快.