Standard Code Library

Your TeamName

Your School

May 12, 2024

Contents

一切的开始

宏定义

- 需要 C++11

```
#include <bits/stdc++.h>
using namespace std;
using LL = long long;
#define FOR(i, x, y) for (decay<decltype(y)>::type i = (x), _##i = (y); i < _##i; ++i)
#define FORD(i, x, y) for (decay<decltype(x)>::type i = (x), _##i = (y); i > _##i; --i)
#ifdef zerol
#define dbg(x...) do { cout << "\033[32;1m" << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<template<typename...> class T, typename t, typename... A>
void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
template<typename T, typename... A>
void err(T a, A... x) { cout << a << ' '; err(x...); }
#else
#define dbg(...)
#endif
// ------------------------------------------------------------------------------
```

数据结构

ST 表

```
struct ST{
    int n;
    std::vector<array<int,21>> st;
    ST(int n):n(n),st(n + 1) {}
    void init(vector<int>& a){
        for(int i = 1;i <= n;i ++)st[i][0] = a[i - 1];
        for(int j = 1;j <= 18;j ++){
            for(int i = 1;i + (1 << j) <= n + 1;i ++){
                st[i][j] = max(st[i][j - 1],st[i + (1 << (j - 1))][j - 1]);
            }
        }
    }
    int rmq(int l,int r){
        int j = log(r - l + 1)/log(2);
        return max(st[l][j],st[r - (1 << j) + 1][j]);
    }
};
```

线段树

```
struct SegTree {
    int l, r;
    SegTree *ls, *rs;
    ll sum;
    ll plus;
    SegTree (const int L, const int R) : l(L), r(R) {
        plus = 0;
        if (L == R) {
            /*Initial*/
            ls = rs = nullptr;
        } else {
            int M = (L + R) >> 1;
            ls = new SegTree (L, M);
            rs = new SegTree (M + 1, R);
            pushup();
        }
    }
    void pushup() {
        sum = ls->sum + rs->sum;
        //  std::cerr << "AAA" << l << ' ' << r << ' ' << sum;
    }
    void make_tag(long long w) {
        sum += (r - l + 1) * w;
        plus += w;
    }
    void pushdown() {
        if (plus == 0) return;
        ls->make_tag(plus);
        rs->make_tag(plus);
        plus = 0;
    }
    void upd(const int L, const int R, const int w) {
        if ((L > r) || (l > R)) return;
        if ((L <= l) && (r <= R)) {
            make_tag(w);
        } else {
            pushdown();
            ls->upd(L, R, w);
            rs->upd(L, R, w);
            pushup();
        }
    }
};
```

树状数组

```
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;
    Fenwick(int n) : n(n), a(n) {}
    void add(int x, T v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] += v;
        }
    }
    T sum(int x) {
        T ans = 0;
        for (int i = x; i > 0; i -= i & -i) {
            ans += a[i - 1];
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }
    int kth(T k) {
        int x = 0;
        // 先从高位开始取，如果当前这一位可以取，那么就考虑下一位是取 1 还是 0
        // 到最后找到的就是最大的那个 pos 并且对应的 <=x 的
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i - 1]) {
                x += i;
                k -= a[x - 1];
            }
        }
        return x;
    }//树状数组上倍增本质上是通过倍增来快速找出对应的区间
};
```

DSU

```
struct DSU {
    std::vector<int> f, siz;
    DSU(int n) : f(n), siz(n, 1) { std::iota(f.begin(), f.end(), 0); }
    int leader(int x) {
        while (x != f[x]) x = f[x] = f[f[x]];
        return x;
    }
    bool same(int x, int y) { return leader(x) == leader(y); }
    bool merge(int x, int y) {
        x = leader(x);
        y = leader(y);
        if (x == y) return false;
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }
    int size(int x) { return siz[leader(x)]; }
};
```

Splay

```
struct Node {
  int v, sz, sm;
  Node *ch[2], *fa;

  Node(const int V, Node *const f) : v(V), sz(1), sm(1), fa(f) {
    ch[0] = ch[1] = nullptr;
  }

  inline int GetRela(const int x) { return (v == x) ? -1 : (x > v); }

  void pushup() { sm = (ch[0] ? ch[0]->sm : 0) + (ch[1] ? ch[1]->sm : 0) + sz; }

  inline void rotate(const int x) {
    auto nrt = ch[x];
    ch[x] = nrt->ch[x ^ 1];
    nrt->ch[x ^ 1] = this;
    if (ch[x]) ch[x]->fa = this;
    nrt->fa = fa; fa = nrt;
    if (nrt->fa) nrt->fa->ch[nrt->fa->GetRela(nrt->v)] = nrt;
    pushup(); nrt->pushup();
  }

  void splay(const Node *p) {
    while (fa != p) {
      auto pa = fa->fa;
      if (pa == p) {
        fa->rotate(fa->GetRela(v));
      } else {
        int k1 = fa->GetRela(v), k2 = pa->GetRela(fa->v);
        if (k1 == k2) {
          pa->rotate(k1);
          fa->rotate(k1);
        } else {
          fa->rotate(k1);
          fa->rotate(k2);
        }
      }
    }
  }
};
```

LCT

```
struct Node {
  int v, s;
  bool tag;
  Node *ch[2], *fa;

  inline void maketag() {
    tag = !tag;
    std::swap(ch[0], ch[1]);
  }
  inline void pushup() {
    s = v;
    for (auto u : ch) if (u != nullptr) {
      s ^= u->s;
    }
  }
  inline void pushdown() {
    if (tag) {
      for (auto u : ch) if (u != nullptr) {
        u->maketag();
      }
      tag = false;
    }
  }

  inline int Getson() { return fa->ch[1] == this; }

  inline bool IsRoot() { return (fa == nullptr) || (fa->ch[Getson()] != this); }

  void rotate(const int x) {
    auto nt = ch[x];
    ch[x] = nt->ch[x ^ 1];
    nt->ch[x ^ 1] = this;
    if (ch[x]) ch[x]->fa = this;
    nt->fa = fa;
    if (!IsRoot()) { fa->ch[Getson()] = nt; }
    fa = nt;
    pushup(); nt->pushup();
  }

  void splay() {
    static Node* stk[maxn];
    int top = 0;
    stk[++top] = this;
    for (auto u = this; !u->IsRoot(); stk[++top] = u = u->fa);
    while (top) stk[top--]->pushdown();
    while (!IsRoot()) {
      if (fa->IsRoot()) {
        fa->rotate(Getson());
      } else {
        auto pa = fa->fa;
        int l1 = Getson(), l2 = fa->Getson();
        if (l1 == l2) {
          pa->rotate(l2);
          fa->rotate(l1);
        } else {
          fa->rotate(l1);
          fa->rotate(l2);
        }
      }
    }
  }
};
Node *node[maxn], Mem[maxn];

void Cut(const int x, const int y);
void Link(const int x, const int y);
void Query(const int x, const int y);
void Update(const int x, const int y);

void access(Node *u) {
  for (Node *v = nullptr; u; u = (v = u)->fa) {
    u->splay();
    u->ch[1] = v; u->pushup();
  }
}

void makeroot(Node *const u) {
  access(u);
  u->splay();
  u->maketag();
}

void Query(const int x, const int y) {
  auto u = node[x], v = node[y];
  makeroot(u);
  access(v);
  v->splay();
  qw(v->s, '\n');
```

```
}

void Link(const int x, const int y) {
  auto u = node[x], v = node[y];
  makeroot(u);
  access(v); v->splay();
  if (u->IsRoot() == false) return;
  u->fa = v;
}

void Cut(const int x, const int y) {
  auto u = node[x], v = node[y];
  makeroot(u); access(v); u->splay();
  if ((u->ch[1] != v) || (v->ch[0] != nullptr)) return;
  u->ch[1] = v->fa = nullptr;
  u->pushup();
}

// w[x] -> y
void Update(const int x, const int y) {
  auto u = node[x];
  u->splay();
  u->s ^= u->v;
  u->s ^= (u->v = a[x] = y);
}
```

扫描线

```
//二维数点
struct Segment{
    int l,r,h,add;
    bool operator <(const Segment a)const{
        return h < a.h;
    }
};
struct SegTree {
    int l, r;
    SegTree *ls, *rs;
    int mn,len;
    int plus;
    SegTree (const int L, const int R) : l(L), r(R) {
        plus = 0;len = 0;
        if (L == R) {
            ls = rs = nullptr;
        } else {
            int M = (L + R) >> 1;
            ls = new SegTree (L, M);
            rs = new SegTree (M + 1, R);
            pushup();
        }
    }
    void pushup() {
        if(plus) len = r - l + 1;
        else if(l == r)len = 0;
        else len = ls->len + rs->len;
    }
    void make_tag(int w) {
        plus += w;
    }
    void pushdown() {
        if (plus == 0) return;
        ls->make_tag(plus);
        rs->make_tag(plus);
        plus = 0;
    }
    void update(const int L, const int R, const int w) {
        if ((L > r) || (l > R)) {
            return;
        }
        if ((L <= l) && (r <= R)) {
            make_tag(w);
```

```cpp
            pushup();
            return ;
        } else {
            ls->update(L, R, w);
            rs->update(L, R, w);
            pushup();
        }
    }
};
//矩形面积并
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;
const double eps = 1e-8;
const int maxn = 2e5 + 7;
std::vector<int> x;
struct Segment{
    int l,r,h,add;
    bool operator <(const Segment a)const{
        return h < a.h;
    }
};
struct SegTree {
    int l, r;
    SegTree *ls, *rs;
    int mn,len;
    int plus;
    SegTree (const int L, const int R) : l(L), r(R) {
        plus = 0;len = 0;
        if (L == R) {
            ls = rs = nullptr;
        } else {
            int M = (L + R) >> 1;
            ls = new SegTree (L, M);
            rs = new SegTree (M + 1, R);
            pushup();
        }
    }
    void pushup() {
        if(plus) len = x[r] - x[l - 1];
        else if(l == r)len = 0;
        else len = ls->len + rs->len;
    }
    void make_tag(int w) {
        plus += w;
    }
    void pushdown() {
        if (plus == 0) return;
        ls->make_tag(plus);
        rs->make_tag(plus);
        plus = 0;
    }
    void update(const int L, const int R, const int w) {
        if ((L >= x[r]) || (x[l - 1] >= R)) {
            return;
        }
        if ((L <= x[l - 1]) && (x[r] <= R)) {
            make_tag(w);
            pushup();
            return ;
        } else {
            //pushdown();
            ls->update(L, R, w);
            rs->update(L, R, w);
            pushup();
        }
    }
};
int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    vector<Segment> s;
    int n;
    cin >> n;
    for(int i = 0;i < n;i ++){
        int xa,ya,xb,yb;
        cin >> xa >> ya >> xb >> yb;
        x.push_back(xa);
        x.push_back(xb);
        s.push_back({xa,xb,ya,1});
        s.push_back({xa,xb,yb,-1});
    }
    sort(s.begin(),s.end());
    sort(x.begin(),x.end());
    x.erase(unique(x.begin(),x.end()),x.end());
    int N = x.size();
    SegTree Seg(1,N - 1);
    ll ans = 0;
    if(s.size()){
        Seg.update(s[0].l,s[0].r,s[0].add);
        for(int i = 1;i < s.size();i ++){
            ans += 1ll * Seg.len * (s[i].h - s[i - 1].h);
            Seg.update(s[i].l,s[i].r,s[i].add);
        }
    }
    cout << ans << "\n";
    return 0;
}
```

Seg beats

本质上是维护了两棵线段树，A 树维护区间内最大值产生的贡献，B 树维护剩下树的贡献。注意 A 树某节点的孩子不一定全部能贡献到该节点，因为孩子的最大值不一定是父亲的最大值。所以要注意下传标记时，A 树的孩子下传的可能是 B 的标记。

beats 的部分是，每次让序列里每个数对另一个数 V 取 min，则直接暴力递归到 inRange 且 B 的最大值小于 V 的那些节点上，转化成对 A 那个节点的区间加法（加上 $V - val_A$）即可。这么做的均摊复杂度是 $O(\log n)$。

做区间历史最大值的方法是，维护两个标记 x, y，x 是真正的加标记，y 是 x 在上次下传结束并清零后的历史最大值。下传时注意先下传 y 再下传 x。实现历史最值是平凡的，不需要 beats。beats 解决的仅是取 min 的操作。

下面五个操作分别是：区间加，区间对 k 取 min，区间求和，区间最大值，区间历史最大值。

```
#include <array>
#include <iostream>
#include <algorithm>

typedef long long int ll;

const int maxn = 500005;

ll a[maxn];

const ll inf = 0x3f3f3f3f3f3f3f3fll;

struct Node {
  Node *ls, *rs;
  int l, r, maxCnt;
  ll v, add, maxAdd, sum, maxV, maxHistory;

  Node(const int L, const int R) :
      ls(nullptr), rs(nullptr), l(L), r(R), maxCnt(0),
      v(0), add(0), maxAdd(0), sum(0), maxV(-inf), maxHistory(-inf) {}

  inline bool inRange(const int L, const int R) {
    return L <= l && r <= R;
  }
  inline bool outRange(const int L, const int R) {
    return l > R || L > r;
  }
```

```cpp
  void addVal(const ll t, int len) {
    add += t;
    sum += len * t;
    maxV += t;
  }

  void makeAdd(const ll t, int len) {
    addVal(t, len);
    maxHistory = std::max(maxHistory, maxV);
    maxAdd = std::max(maxAdd, add);
  }
};

void pushup(Node *x, Node *y) {
  y->maxV = std::max(y->ls->maxV, y->rs->maxV);
  y->sum = y->ls->sum + y->rs->sum;
  y->maxHistory = std::max({y->maxHistory, y->ls->maxHistory, y->rs->maxHistory});
  if (x->ls->maxV != x->rs->maxV) {
    bool flag = x->ls->maxV < x->rs->maxV;
    if (flag) std::swap(x->ls, x->rs);
    x->maxV = x->ls->maxV;
    x->maxCnt = x->ls->maxCnt;
    y->maxV = std::max(y->maxV, x->rs->maxV);
    y->sum += x->rs->sum;
    x->sum = x->ls->sum;
    if (flag) std::swap(x->ls, x->rs);
  } else {
    x->maxCnt = x->ls->maxCnt + x->rs->maxCnt;
    x->sum = x->ls->sum + x->rs->sum;
    x->maxV = x->ls->maxV;
  }
  x->maxHistory = std::max({x->ls->maxHistory, x->rs->maxHistory, x->maxHistory, y->maxHistory});
}

void New(Node *&u1, Node *&u2, int L, int R) {
  u1 = new Node(L, R);
  u2 = new Node(L, R);
  if (L == R) {
    u1->v = u1->sum = u1->maxV = u1->maxHistory = a[L];
    u1->maxCnt = 1;
  } else {
    int M = (L + R) >> 1;
    New(u1->ls, u2->ls, L, M);
    New(u1->rs, u2->rs, M + 1, R);
    pushup(u1, u2);
  }
}

void pushdown(Node *x, Node *y) {
  ll val = std::max(x->ls->maxV, x->rs->maxV);
  std::array<Node*, 2> aim({y, x});
  Node *curl = aim[x->ls->maxV == val], *curr = aim[x->rs->maxV == val];
  x->ls->maxAdd = std::max(x->ls->maxAdd, x->ls->add + curl->maxAdd);
  x->ls->maxHistory = std::max(x->ls->maxHistory, x->ls->maxV + curl->maxAdd);
  x->ls->addVal(curl->add, x->ls->maxCnt);
  x->rs->maxAdd = std::max(x->rs->maxAdd, x->rs->add + curr->maxAdd);
  x->rs->maxHistory = std::max(x->rs->maxHistory, x->rs->maxV + curr->maxAdd);
  x->rs->addVal(curr->add, x->rs->maxCnt);
  y->ls->maxAdd = std::max(y->ls->maxAdd, y->ls->add + y->maxAdd);
  y->rs->maxAdd = std::max(y->rs->maxAdd, y->rs->add + y->maxAdd);
  y->ls->addVal(y->add, x->ls->r - x->ls->l + 1 - x->ls->maxCnt);
  y->rs->addVal(y->add, x->rs->r - x->rs->l + 1 - x->rs->maxCnt);
  x->add = y->add = x->maxAdd = y->maxAdd = 0;
}

void addV(Node *x, Node *y, int L, int R, ll k) {
  if (x->inRange(L, R)) {
    x->makeAdd(k, x->maxCnt);
    y->makeAdd(k, x->r - x->l + 1 - x->maxCnt);
  } else if (!x->outRange(L, R)) {
    pushdown(x, y);
    addV(x->ls, y->ls, L, R, k);
    addV(x->rs, y->rs, L, R, k);
    pushup(x, y);
  }
}

std::array<ll, 3> qry(Node *x, Node *y, const int L, const int R) {
  if (x->inRange(L, R)) return {x->sum + y->sum * ((x->r - x->l + 1) != x->maxCnt), x->maxV, x->maxHistory};
  else if (x->outRange(L, R)) return {0, -inf, -inf};
  else {
    pushdown(x, y);
    auto A = qry(x->ls, y->ls, L, R), B = qry(x->rs, y->rs, L, R);
    return {A[0] + B[0], std::max(A[1], B[1]), std::max(A[2], B[2])};
  }
}

void minV(Node *x, Node *y, const int L, const int R, int k) {
  if (x->maxV <= k) return;
  if (x->inRange(L, R) && y->maxV < k) {
    ll delta = k - x->maxV;
    x->makeAdd(delta, x->maxCnt);
  } else if (!x->outRange(L, R)) {
    pushdown(x, y);
    minV(x->ls, y->ls, L, R, k);
    minV(x->rs, y->rs, L, R, k);
    pushup(x, y);
  }
}

int main() {
  std::ios::sync_with_stdio(false);
  std::cin.tie(nullptr);
  int n, m;
  std::cin >> n >> m;
  for (int i = 1; i <= n; ++i) std::cin >> a[i];
  Node *rot1, *rot2;
  New(rot1, rot2, 1, n);
  for (int op, l, r; m; --m) {
    std::cin >> op >> l >> r;
    if (op == 1) {
      std::cin >> op;
      addV(rot1, rot2, l, r, op);
    } else if (op == 2) {
      std::cin >> op;
      minV(rot1, rot2, l, r, op);
    } else {
      std::cout << qry(rot1, rot2, l, r)[op - 3] << '\n';
    }
  }
}
```

珂朵莉树

```
auto getPos(int pos) {
  return --s.upper_bound({pos + 1, 0, 0});
}

void split(int pos) {
  auto it = getPos(pos);
  auto [l, r, v] = *it;
  s.erase(it);
  if (pos > l) s.insert({l, pos - 1, v});
  s.insert({pos, r, v});
}

void add(int l, int r, int v) {
  split(l); split(r + 1);
  for (auto x = getPos(l), y = getPos(r + 1); x != y; ++x) {
    x->v += v;
  }
```

```cpp
}

void upd(int l, int r, int v) {
  split(l); split(r + 1);
  s.erase(getPos(l), getPos(r + 1));
  s.insert({l, r, v});
}
```

`getPos(pos)`：找到 pos 所在的迭代器
`split(pos)`：把 pos 所在的迭代器区间 [l, r] 分成 [l, pos - 1] 和 [pos, r] 两个

李超树

插入线段 kx + b 求某点最值

```cpp
constexpr long long INF = 1'000'000'000'000'000;
constexpr int C = 100'000;
struct Line {
    int k;
    long long b;
    Line(int k, long long b) : k(k), b(b) {}
};
long long f(const Line &line, int x) {
    return 1LL * line.k * x + line.b;
}
struct Node {
    Node *lc, *rc;
    Line line;
    Node(const Line &line) : lc(nullptr), rc(nullptr), line(line) {}
};
void modify(Node *&p, int l, int r, Line line) {
    if (p == nullptr) {
        p = new Node(line);
        return;
    }
    int m = (l + r) / 2;
    bool le = f(p -> line, l) < f(line, l);
    bool mi = f(p -> line, m) < f(line, m);
    if (!mi)
        std::swap(p -> line, line);
    if (r - l == 1)
        return;
    if (le != mi) {
        modify(p -> lc, l, m, line);
    } else {
        modify(p -> rc, m, r, line);
    }
}
Node *merge(Node *p, Node *q, int l, int r) {
    if (p == nullptr)
        return q;
    if (q == nullptr)
        return p;
    int m = (l + r) / 2;
    p -> lc = merge(p -> lc, q -> lc, l, m);
    p -> rc = merge(p -> rc, q -> rc, m, r);
    modify(p, l, r, q -> line);
    return p;
}
long long query(Node *p, int l, int r, int x) {
    if (p == nullptr)
        return INF;
    long long ans = f(p -> line, x);
    if (r - l == 1)
        return ans;
    int m = (l + r) / 2;
    if (x < m) {
        return std::min(ans, query(p -> lc, l, m, x));
    } else {
        return std::min(ans, query(p -> rc, m, r, x));
    }
}
```

动态维护凸壳

```
/**
 * Author: Simon Lindholm
 * Date: 2017-04-20
 * License: CC0
 * Source: own work
 * Description: Container where you can add lines of the form kx+m, and query maximum values at points x.
 *  Useful for dynamic programming.
 * Time: O(\log N)
 * Status: tested
 */

struct Line {
  mutable ll k, m, p;
  bool operator<(const Line &o) const { return k < o.k; }
  bool operator<(ll x) const { return p < x; }
};

struct LineContainer: multiset<Line, less<>> {
  const ll inf = LLONG_MAX;
  ll val_offset = 0;
  void offset(ll x) {
    val_offset += x;//整体加
  }
  ll div(ll a, ll b) {
    return a / b - ((a^b) < 0 && a%b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) {
      x->p = inf;
      return 0;
    }
    if (x->k == y->k) {
      x->p = (x->m > y->m)? inf: -inf;
    } else {
      x->p = div(y->m - x->m, x->k - y->k);
    }
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m - val_offset, 0}), y = z++, x = y;//这里加减看情况
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)->p >= y->p) isect(x, erase(y));
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m + val_offset;
  }
};

LineContainer* merge(LineContainer *S, LineContainer *T) {
  if (S->size() > T->size())
    swap(S, T);
  for (auto l: *S) {
    T->add(l.k, l.m + S->val_offset);
  }
  return T;
}
```

TODO

线段树合并和分裂

计算几何

二维几何：点与向量

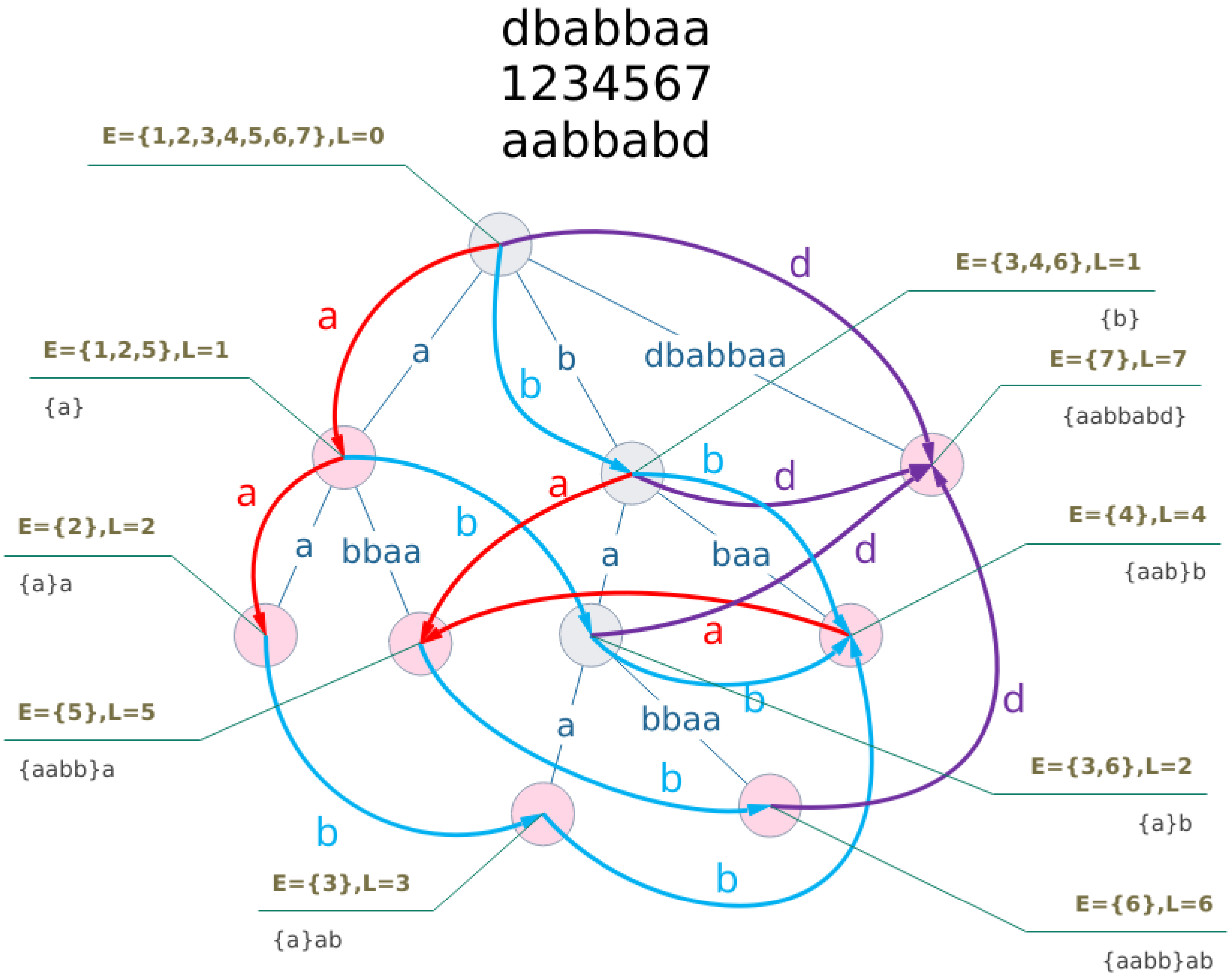
```
#define y1 yy1
#define nxt(i) ((i + 1) % s.size())
typedef double LD;
const LD PI = 3.14159265358979323846;
const LD eps = 1E-10;
int sgn(LD x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); }
struct L;
struct P;
typedef P V;
struct P {
    LD x, y;
    explicit P(LD x = 0, LD y = 0): x(x), y(y) {}
    explicit P(const L& l);
};
struct L {
    P s, t;
    L() {}
    L(P s, P t): s(s), t(t) {}
};

P operator + (const P& a, const P& b) { return P(a.x + b.x, a.y + b.y); }
P operator - (const P& a, const P& b) { return P(a.x - b.x, a.y - b.y); }
P operator * (const P& a, LD k) { return P(a.x * k, a.y * k); }
P operator / (const P& a, LD k) { return P(a.x / k, a.y / k); }
inline bool operator < (const P& a, const P& b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const P& a, const P& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
P::P(const L& l) { *this = l.t - l.s; }
ostream &operator << (ostream &os, const P &p) {
    return (os << "(" << p.x << "," << p.y << ")");
}
istream &operator >> (istream &is, P &p) {
    return (is >> p.x >> p.y);
}

LD dist(const P& p) { return sqrt(p.x * p.x + p.y * p.y); }
LD dot(const V& a, const V& b) { return a.x * b.x + a.y * b.y; }
LD det(const V& a, const V& b) { return a.x * b.y - a.y * b.x; }
LD cross(const P& s, const P& t, const P& o = P()) { return det(s - o, t - o); }
// --------------------------------------------
```

字符串

后缀自动机



杂项

STL

- copy

```
template <class InputIterator, class OutputIterator>
  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);
```