# 线段树

## 1.单点更新

**说明**

单点更新，区间求和（你问我单点求和？？你就不会把区间长度设为0啊？）

* sum[]为线段树，需要开辟四倍的元素数量的空间。
* build()为建树操作
* update()为更新操作
* query()为查询操作

时间复杂度：O(nlogn)

**使用方法**

1. build(1, n); 建立一个叶子节点为n个的线段树
2. update(pos, val, 1, n); 更新树中下标为pos的叶子节点值增加val
3. query(l, r, 1, n); 查询[l ,r]区间值之和

**Tips**

* 请注意update的目的是**增减**还是**替换**，根据情况修改update函数和pushup函数
* 建出来的树为空树，默认每个点值都为0，需要自行将值update上去，或者修改build中sum[rt]=0;为输入操作scanf("%d",sum+rt);

**模版**

|  |
| --- |
| // 有注释版  const int maxn=2005+5;  #define lson l,m,rt<<1 //预定子左树  #define rson m+1,r,rt<<1|1 //预定右子树  int sum[maxn<<2];//表示节点，需要开到最大区间的四倍  void pushup(int rt){  //对于编号为rt的节点，他的左右节点分别为rt<<1和rt<<1|1  sum[rt]=sum[rt<<1]+sum[rt<<1|1];  }  //造树  void build(int l,int r,int rt=1){  //建树操作，生成一个区间为l~r的完全二叉树  //如果到底，则线段长度为0，表示一个点，输入该点的值  if (l==r) {  sum[rt]=0;  return;  }  //准备子树  int m=(l+r)>>1;  //对当前节点建立子树  build(lson);  build(rson);  //由底向上求和  pushup(rt);  }  //更新点和包含点的枝  void update(int pos,int val,int l,int r,int rt=1){  //pos为更新的位置 val为增加的值，正则加，负则减  //l r为区间的两个端点值  //触底，为一个点的时候，该节点值更新  if (l==r) {  sum[rt]+=val;  return;  }  int m = ( l + r ) >> 1;  if (pos<=m) //pos在左子树的情况下，对左子树进行递归  update(pos, val, lson);  else //pos在右子树的情况下，对右子树进行递归  update(pos, val, rson);  //更新包含该点的一系列区间的值  pushup(rt);  }  //查询点或区间  int query(int L,int R,int l,int r,int rt=1){  // L~R为被查询子区间 l~r为“当前”树的全区间  if (L<=l&&r<=R) //子区间包含“当前”树全区间  return sum[rt]; //返回该节点包含的值  int m=(l+r)>>1;  int res=0;  if (L<=m) //左端点在左子树内  res+=query(L, R, lson);  if (R>m) //右端点在右子树内  res+=query(L, R, rson);  return res;  } |

## 2.区间更新

**说明**

区间更新，区间求和（你问我单点求和？？你就不会把区间长度设为0啊？）

* sum[]为线段树，需要开辟四倍的元素数量的空间。
* build()为建树操作
* update()为更新操作
* query()为查询操作

时间复杂度：O(nlogn)

**使用方法**

1. build(1, n); 建立一个叶子节点为n个的线段树
2. update(l, r, val, 1, n); 更新线段树中[l, r]区间每个值都增加val
3. query(l, r, 1, n); 查询[l ,r]区间值之和

**Tips**

* 请注意update的目的是**增减**还是**替换**，根据情况修改update函数和pushup函数
* 建出来的树为空树，默认每个点值都为0，需要自行将值update上去，或者修改build中sum[rt]=0;为输入操作scanf("%d",sum+rt);

**模版**

|  |
| --- |
| #define lson l,m,rt<<1  #define rson m+1,r,rt<<1|1  const int maxn = 100005;  int add[maxn<<2],sum[maxn<<2];  void PushUp(int rt)  {  sum[rt]=sum[rt<<1]+sum[rt<<1|1];  }  void PushDown(int rt,int m)  {  if (add[rt])  {  add[rt<<1] += add[rt];  add[rt<<1|1] += add[rt];  sum[rt<<1] += add[rt] \* (m - (m >> 1));  sum[rt<<1|1] += add[rt] \* (m >> 1);  add[rt] = 0;  }  }  void build(int l,int r,int rt=1)  {  add[rt] = 0;  if (l == r)  {  sum[rt]=0;  return ;  }  int m = (l + r) >> 1;  build(lson);  build(rson);  PushUp(rt);  }  void update(int L,int R,int c,int l,int r,int rt=1)  {  if (L <= l && r <= R)  {  add[rt] += c;  sum[rt] += c \* (r - l + 1);  return ;  }  PushDown(rt , r - l + 1);  int m = (l + r) >> 1;  if (L <= m) update(L , R , c , lson);  if (m < R) update(L , R , c , rson);  PushUp(rt);  }  int query(int L,int R,int l,int r,int rt=1)  {  if (L <= l && r <= R)  {  return sum[rt];  }  PushDown(rt , r - l + 1);  int m = (l + r) >> 1;  int ret = 0;  if (L <= m) ret += query(L , R , lson);  if (m < R) ret += query(L , R , rson);  return ret;  } |

## 3.RMQ

**说明**

RMQ：Range Minimum(Maximum) Query

* sum[]为线段树，需要开辟四倍的元素数量的空间。
* build()为建树操作
* update()为更新操作
* query()为查询操作

**使用方法**

1. 根据情况修改RMQ的宏定义
2. build(1, n); 建立一个叶子节点为n个的线段树
3. update(pos, val, 1, n); 修改树中下标为pos的叶子节点值为val
4. query(l, r, 1, n); 查询[l ,r]区间中的RMQ

**Tips**

* 建出来的树为空树，默认每个点值都为0，需要自行将值update上去，或者修改build中sum[rt]=0;为输入操作scanf("%d",sum+rt);
* RMQ为宏定义，请根据情况自行修改为max或者min，对应修改query中的res为-INF或者INF

|  |
| --- |
| const int maxn=2005+5;  #define RMQ max  #define lson l,m,rt<<1  #define rson m+1,r,rt<<1|1  int sum[maxn<<2]={};  void pushup(int rt){  sum[rt]=RMQ(sum[rt<<1],sum[rt<<1|1]);  }  void build(int l,int r,int rt=1){  if (l==r){  sum[rt]=0;  return;  }  int m=(l+r)>>1;  build(lson);  build(rson);  pushup(rt);  }  void update(int pos,int val,int l,int r,int rt=1){  if (l==r) {  sum[rt]=val;  return;  }  int m=(l+r)>>1;  if (pos<=m) update(pos, val, lson);  else update(pos, val, rson);  pushup(rt);  }  int query(int L,int R,int l,int r,int rt=1){  if (L<=l&&r<=R) return sum[rt];  int m=(l+r)>>1;  int res=-INF; //防负数的坑  if (L<=m) res=RMQ(res,query(L, R, lson));  if (R>m) res=RMQ(res,query(L, R, rson));  return res;  } |

18校赛F题意：给你一个字符串，各两个操作:   
ch=0,[l,r]降序   
ch=1,[l,r]升序

|  |
| --- |
| #include <cstdio>  #include <cstring>  #include <algorithm>  #include <cstdlib>  #define ls (o<<1)  #define rs (o<<1|1)  #define lson ls,L,M  #define rson rs,M+1,R  using namespace std;  typedef long long ll;  const int N = 100005;  const int sigma = 26;  int sumv[N<<2][sigma], setv[N<<2]; //setv表示当前区间被覆盖的字母  int cnt[26]; //每个坐标在当前区间出现的次数  char str[N];  void pushUp(int o) {  for(int i = 0; i < sigma; i++) {  sumv[o][i] = sumv[ls][i] + sumv[rs][i];  }  }  void initNode(int o) {  memset(sumv[o], 0, sizeof(sumv[o]));  }  void pushDown(int o, int L, int R) {  int M = (L+R)/2;  if(setv[o] != -1) {  setv[ls] = setv[rs] = setv[o];  initNode(ls);  sumv[ls][setv[o]] = (M-L+1);  initNode(rs);  sumv[rs][setv[o]] = (R-M);  setv[o] = -1;  }  }  void build(int o, int L, int R) {  if(L == R) {  initNode(o);  sumv[o][str[L]-'a'] = 1;  setv[o] = str[L]-'a';  return ;  }  setv[o] = -1;  int M = (L+R)/2;  build(lson);  build(rson);  pushUp(o);  }  int query(int o, int L, int R, int ql, int qr, int val) {  if(ql <= L && R <= qr)  return sumv[o][val];  pushDown(o, L, R);  int M = (L+R)/2, ret = 0;  if(ql <= M) ret += query(lson, ql, qr, val);  if(qr > M) ret += query(rson, ql, qr, val);  return ret;  }  void modify(int o, int L, int R, int ql, int qr, int val) {  if(ql <= L && R <= qr) {  for(int i = 0; i < sigma; i++)  sumv[o][i] = 0;  setv[o] = val;  sumv[o][setv[o]] = (R-L+1);  return ;  }  pushDown(o, L, R);  int M = (L+R)/2;  if(ql <= M) modify(lson, ql, qr, val);  if(qr > M) modify(rson, ql, qr, val);  pushUp(o);  }  void getStr(int o, int L, int R) {  if(L == R) {  printf("%c", (char)('a'+setv[o]));  return ;  }  pushDown(o, L, R);  int M = (L+R)/2;  getStr(lson);  getStr(rson);  }  int n, q;  int main() {  while(scanf("%d%d", &n, &q) != EOF) {  scanf("%s", str+1);  build(1, 1, n);  int ql, qr, ch;  while(q--) {  scanf("%d%d%d", &ql, &qr, &ch);  for(int i = 0; i < sigma; i++)  cnt[i] = query(1, 1, n, ql, qr, i);  int pos = ql;  if(ch == 1) {  for(int i = 0; i < sigma; i++) {  if(cnt[i] > 0)  modify(1, 1, n, pos, pos+cnt[i]-1, i);  pos += cnt[i];  }  }else {  for(int i = sigma-1; i >= 0; i--) {  if(cnt[i] > 0)  modify(1, 1, n, pos, pos+cnt[i]-1, i);  pos += cnt[i];  }  }  }  getStr(1, 1, n); puts("");  }  return 0;  } |