Legendgod's Blog

一下子没有什么精神，开一个博客来总结一下，我曾经会过什么东西，同时就当做复习了。

预计会写稍微多一点东西，尽量按照 `oi-wiki` 的结构和自己的理解来。

队列

常常就是用 `queue`, `priority_queue`, `deque`，分别是普通队列，优先队列，双端队列。

1. 优化 Dp，比如说维护一个最大值，就是考虑如果说当前值比之后加入位置的小，显然当前的位置就不用了。每次我们取出队头的最大值即可。

> 优化斜率 Dp，常常说是否要取等的情况，明显这个队列是维护斜率单调的，至于取等的情况，我们考虑斜率相同但是截距不同的情况，所以是需要删除前面的斜率相同的位置。

2. 维护最短路，这个比较显然就不说了。
3. 记住这个东西是先进先出就可以了。

栈

就只有一个 `stack`。

1. 维护最值，还是考虑维护最大值，我们对比一下队列的维护方式。队列可以加入当前不是最大值但是之后最大值删掉之后可能产生贡献的情况，但是栈如果发现不能加入之后就不会再加入了，所以说队列是可以维护区间最值，但是栈更多是来维护全局最值。
2. 后进先出，比如表达式求值。
3. 双栈模拟队列，考虑维护两个栈表示队头和队尾，每次 `push` 的时候放到队尾，每次 `pop` 的时候看队头是否有元素，没有的话就让队尾一直弹出元素到队头，每个元素均摊 $O(1)$ 复杂度是 $O(n)$ 的。

例题： LOJ6515「雅礼集训 2018 Day10」贪玩蓝月

现在我们有若干种事件和询问，如下所示：

- `IF w v`：在前端加入一件特征值为 w 战斗力为 v 的装备
- `IG w v`：在后端加入一件特征值为 w 战斗力为 v 的装备

- `DF`：删除最前端的装备
- `DG`：删除最后端的装备
- `QU l r`：在当前的装备中选取若干装备，他们的和对 p 取模后在 $[l,r]$ 中，使得这些装备的战斗力之和最大。

为了锻炼你的水平，请尽量使用在线做法。

发现是双端队列，本质上就是求 $[\sum w \mod p \in [l,r]] \sum v$，发现每个二元组有一个存活的时间，我们先考虑一下比较简单的情况，比如说让维护一个栈。

离线很显然是线段树分治。

后进先出，考虑进行 Dp 设 $f(i,j)$ 考虑前 i 个数，当前余数是 j，转移是一个背包。

$$f(i,j) \to f(i+1,j), f(i,j) \to f(i+1,(j+v) \mod p)$$

删除呢？我们考虑真正需要关注的就是维护了所有的二元组的答案，既然是栈我们直接删掉这个元素即可。

> 看起来非常得方便。

我们考虑用双栈来模拟队列，同时维护这个 Dp，我们还有一个问题就是考虑两个 Dp，合并 Dp 的复杂度是多少？可以直接进行卷积复杂度是 $O(mp\log p)$，或者暴力进行是 $O(mp^2)$。

考虑算答案的时候进行计算，一个合法的条件是 $f(i,a)+f(j,b)$ 其中 $(a+b) \mod p \in [l,r]$。

考虑对于同一个 a 考虑 b，可以得到 b 的限制，我们是需要取最大值使用单调队列优化即可。

诶，他是双端队列呀！我们常规维护两个栈是直接将一个栈的所有元素放到另外一个上，会发现单次操作复杂度可以到 $O(m)$，我们考虑能否均摊一下， 最明显的想法就是他不是要相互倒来倒去吗？我们两边分别留一半即可。

比如时候一开始有 m 个元素，我们丢一半之后需要花费 $O(\frac{m}{2})$ 的时间删除，所以复杂度是 $O(mp)$ 的。

```cpp
#include <bits/stdc++.h>
using namespace std;
namespace Legendgod {
    namespace Read {
//        #define Fread
        #ifdef Fread
        const int Siz = (1 << 21) + 5;
        char *iS, *iT, buf[Siz];
        #define gc() ( iS == iT ? (iT = (iS = buf) + fread(buf, 1, Siz, stdin), iS == iT ? EOF : *iS ++) : *iS ++ )
        #define getchar gc
        #endif
        template <typename T>
        void r1(T &x) {
            x = 0;
            char c(getchar());
            int f(1);
            for(; !isdigit(c); c = getchar()) if(c == '-') f = -1;
            for(; isdigit(c); c = getchar()) x = (x << 1) + (x << 3) + (c ^ 48);
            x *= f;
        }
        template <typename T, typename...Args>
        void r1(T &x, Args&...arg) {
            r1(x), r1(arg...);
        }
        #undef getchar
    }

using namespace Read;

const int maxn = 5e4 + 5;
const int maxp = 5e2 + 5;
int n, m, mod;
int ln[2], f[2][maxn][maxp];// front back
pair<int, int> as[2][maxn];

void Max(int &a,const int& c) { a < c ? a = c : 0; }

void Update(const pair<int, int>& s, int f[][maxp], int ps) {
    for(int i = 0; i < mod; ++ i) {
        f[ps][i] = f[ps - 1][i];
        int tmp = (i - s.first + mod) % mod;
        if(f[ps - 1][tmp] >= 0)
        Max(f[ps][i], f[ps - 1][tmp] + s.second);
    }
//    printf("f[ps] = %d\n", f[ps][])
}

void push_front(const pair<int, int>& s) { ++ ln[0], as[0][ln[0]] = s, Update(s, f[0], ln[0]); }
```

```
void push_back(const pair<int, int>& s) { ++ ln[1], as[1][ln[1]] = s, Update(s, f[1],
ln[1]); }

void pop_front() {
    if(ln[0]) return -- ln[0], void();
    int mid = (ln[1] + 1) / 2, tmp = ln[1];
    for(int i = mid; i >= 1; -- i) push_front(as[1][i]);
    ln[1] = 0, -- ln[0];
    for(int i = mid + 1; i <= tmp; ++ i) push_back(as[1][i]);
}

void pop_back() {
    if(ln[1]) return -- ln[1], void();
    int mid = (ln[0] + 1) / 2, tmp = ln[0];
    for(int i = mid; i >= 1; -- i) push_back(as[0][i]);
    ln[0] = 0, -- ln[1];
    for(int i = mid + 1; i <= tmp; ++ i) push_front(as[0][i]);
}

int q[maxn], st, ed;

int Query(int l,int r) {
    st = 1, ed = 0;
    const int *const fl = f[0][ln[0]], *const fr = f[1][ln[1]];
    int ans = -1;
    for(int i = r; i >= l; -- i) {
        int x = (i + mod) % mod;
        while(st <= ed && fr[q[ed]] <= fr[x]) -- ed;
        q[++ ed] = x;
    }
    int ps = l;
    for(int i = 0; i < mod; ++ i) {
        if(st <= ed && fl[i] >= 0 && fr[q[st]] >= 0) ans = max(ans, fl[i] +
fr[q[st]]);
        while(st <= ed && (i + q[st]) % mod == r) ++ st;
        -- ps, ps = (ps + mod) % mod;
        while(st <= ed && fr[q[ed]] <= fr[ps]) -- ed;
        q[++ ed] = ps;
    }
    return ans;
}

char s[10];

signed main() {
    int i, j;
    r1(i, m, mod);
    for(i = 1; i < mod; ++ i) f[0][0][i] = f[1][0][i] = -1;
    for(i = 1; i <= m; ++ i) {
        scanf("%s", s + 1);
        int w, v;
```

```
        if(s[1] == 'I') {
            r1(w, v);
            w %= mod;
            if(s[2] == 'F') push_front({w, v});
            else push_back({w, v});
        }
        else if(s[1] == 'D') {
            if(s[2] == 'F') pop_front();
            else pop_back();
        }
        else {
            r1(w, v);
            printf("%d\n", Query(w, v));
        }
    }
    return 0;
}
/*
0
11 10
QU 0 0
QU 1 9
IG 14 7
IF 3 5
QU 0 9
IG 1 8
DF
QU 0 4
IF 1 2
DG
QU 2 9
*/
}


signed main() { return Legendgod::main(), 0; }//
```

链表

有十分优秀的删除复杂度为 $O(1)$，常见的有单向链表，双向链表，循环链表。

> 就是字面意思。

```
int pre[maxn], suf[maxn], vl[maxn], tot(0);
void Del(int x) {// delete the xth position
    suf[pre[x]] = suf[x], pre[suf[x]] = pre[x];
}

void Insert(int x,int c) {// insert a value behind the xth position
    vl[++ tot] = c;
    suf[tot] = suf[x], pre[tot] = x;
    suf[x] = tot, pre[suf[tot]] = tot;
}
```

介绍一下 `list` 的用法，就是和 `deque` 的函数差不多，但是有一点不同：

```
int a[5] = {0, 1, 2, 3, 4};
list<int> Lis(a, a + 5);// prepare
list<int> cLis(Lis);// copy
array<int, 5> arr{0, 1, 2, 3, 4};
list<int> tLis(10, 5);// room 10 element filled with 5
```

哈希表

基本上使用双模数，模数都是质数比如说: $10^9+7, 10^9+9$ 底数为 131 之类的。

但是 `Hash` 总是会产生冲突的，建议使用拉链法(用上文的链表)。

> 你甚至可以使用链式前向星。

并查集

分为路径压缩和按秩合并(启发式合并)，后者是可以撤销的，比如说在可撤销并查集里面使用。

> 只使用一个的复杂度是 $O(n\log n)$，如果一起使用的复杂度是 $O(n\alpha(n))$，那个反阿克曼函数基本是 5。

扩展域并查集

比如说定义边上的权值，之后在合并的时候计算。

例题： 【NOI2015】程序自动分析 - 题目 - Universal Online Judge

考虑离线，最后一起处理不等于的情况即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
namespace Legendgod {
    namespace Read {
//        #define Fread
        #ifdef Fread
        const int Siz = (1 << 21) + 5;
        char *iS, *iT, buf[Siz];
        #define gc() ( iS == iT ? (iT = (iS = buf) + fread(buf, 1, Siz, stdin), iS == iT ? EOF : *iS ++) : *iS ++ )
        #define getchar gc
        #endif
        template <typename T>
        void r1(T &x) {
            x = 0;
            char c(getchar());
            int f(1);
            for(; !isdigit(c); c = getchar()) if(c == '-') f = -1;
            for(; isdigit(c); c = getchar()) x = (x << 1) + (x << 3) + (c ^ 48);
            x *= f;
        }
        template <typename T, typename...Args>
        void r1(T &x, Args&...arg) {
            r1(x), r1(arg...);
        }
        #undef getchar
    }

using namespace Read;

const int maxn = 2e5 + 5;
int n, m, tmp[maxn], tot(0);
struct Quer {
    int l, r, opt;
    int operator < (const Quer &z) const {
        return opt > z.opt;
    }
}q[maxn];

int fa[maxn];

int getfa(int x) { return x == fa[x] ? x : fa[x] = getfa(fa[x]); }
void merge(int u,int v) {
    u = getfa(u), v = getfa(v);
    if(u != v) fa[u] = v;
}

void Solve() {
    int i, j;
    r1(n), tot = 0;
```

```
    for(i = 1; i <= n; ++ i) {
        r1(q[i].l, q[i].r, q[i].opt);
        tmp[++ tot] = q[i].l, tmp[++ tot] = q[i].r;
    }
    sort(tmp + 1, tmp + tot + 1);
    tot = unique(tmp + 1, tmp + tot + 1) - tmp - 1;
    for(i = 1; i <= tot; ++ i) fa[i] = i;
    sort(q + 1, q + n + 1);
    for(i = 1; i <= n; ++ i) {
        q[i].l = lower_bound(tmp + 1, tmp + tot + 1, q[i].l) - tmp;
        q[i].r = lower_bound(tmp + 1, tmp + tot + 1, q[i].r) - tmp;
        if(q[i].opt == 1) {
            merge(q[i].l, q[i].r);
        }
        else {
            int x = getfa(q[i].l), y = getfa(q[i].r);
//            printf("%d %d\n", x, y);
            if(x == y) return puts("NO"), void();
        }
    }
    puts("YES");
}

signed main() {
    int i, j, T(1);
    r1(T); while(T --) Solve();
    return 0;
}

}


signed main() { return Legendgod::main(), 0; }//
```

带权并查集

[NOI2002] 银河英雄传说 - 洛谷

题意简述：

两个操作：

1. 合并两个位置。
2. 求两个位置的距离。

> 说实话感觉这个东西可以直接建树之后倍增做，我们考虑在线的情况，可以写一个 Lct 直接求距离 · · ·

考虑使用路径压缩保证复杂度，设一个数组 $\mathtt{num}[\mathtt{x}]$ 表示到根节点的距离，每次查找父亲的时候更新 $\mathtt{num}[\mathtt{x}]+=\mathtt{num}[\mathtt{fa}[\mathtt{x}]]$，合并的时候更新到新的父亲的距离就是新父亲的集合大小。

```cpp
#include <bits/stdc++.h>
using namespace std;
namespace Legendgod {
    namespace Read {
//        #define Fread
        #ifdef Fread
        const int Siz = (1 << 21) + 5;
        char *iS, *iT, buf[Siz];
        #define gc() ( iS == iT ? (iT = (iS = buf) + fread(buf, 1, Siz, stdin), iS == iT ? EOF : *iS ++) : *iS ++ )
        #define getchar gc
        #endif
        template <typename T>
        void r1(T &x) {
            x = 0;
            char c(getchar());
            int f(1);
            for(; !isdigit(c); c = getchar()) if(c == '-') f = -1;
            for(; isdigit(c); c = getchar()) x = (x << 1) + (x << 3) + (c ^ 48);
            x *= f;
        }
        template <typename T, typename...Args>
        void r1(T &x, Args&...arg) {
            r1(x), r1(arg...);
        }
        #undef getchar
    }

using namespace Read;

const int maxn = 3e4 + 5;
const int N = 3e4;
int n, m, fa[maxn], num[maxn], siz[maxn];

int getfa(int x) {
    if(x == fa[x]) return x;
    int tmp = getfa(fa[x]);
    num[x] += num[fa[x]];
    return fa[x] = tmp;
}

void merge(int u,int v) {
    u = getfa(u), v = getfa(v);
    if(u == v) return ;
    num[u] = siz[v];
    siz[v] += siz[u];
    fa[u] = v;
}

int Ask(int u,int v) {
```

```
    if(getfa(u) != getfa(v)) return -1;
    if(u == v) return 0;
    return abs(num[u] - num[v]) - 1;
}

char s[10];

signed main() {
    int i, j;
    for(i = 1; i <= N; ++ i) fa[i] = i, num[i] = 0, siz[i] = 1;
    r1(n);
    for(i = 1; i <= n; ++ i) {
        scanf("%s", s + 1);
        int x, y; r1(x, y);
        if(s[1] == 'M') merge(x, y);
        else printf("%d\n", Ask(x, y));
    }
    return 0;
}
/*
4
M 1 2
M 2 3
M 3 4
C 1 4
*/

}


signed main() { return Legendgod::main(), 0; }//
```

二叉堆

> 之前优先队列已经讲过了，我们就直接讲二叉堆了。

维护一个完全二叉树，满足父亲节点的权值比儿子权值大（小就是小根堆了）。

> 如果再加上满足 `bst` 在序列上的性质，就是笛卡尔树了。

1. **`Insert`**，从最右边的儿子进行插入，每次和父亲节点进行调整即可。
2. **`Delete`**，删除父亲不是很好做，考虑将父亲移动到叶子节点进行删除。考虑让父亲节点和叶子节点交换之后删除父亲，让叶子结点从上到下进行调整即可。
3. 增加某个点的权值，直接向上调整即可。

4. 减小某个点的权值，选不合法儿子最大的一个和自己交换即可，一直做。

ST 表

> 其实这个东西应该是我认为我学的第一个数据结构。

可以 $O(n\log n\times x),O(x)$ 查询，其中 x 表示合并信息的复杂度，我们常见的维护最值及其位置是可以 $O(n\log n)-O(1)$ 的。

猫树

其实和 ST 表差不多，但是可以维护的东西本质上更多一些，比如说可以维护最大子段和之类的，复杂度和上述相同。

考虑和线段树一线的构建，发现当前位置的深度就是 $\log_2 p+1$。

> 根节点深度为 1。

对于这个东西还有一个很强的性质就是对于两点 x,y 其树上 Lca 的位置就是二进制下的最长公共前缀。这个东西可以直接通过异或求出除了最长公共前缀的部分。

```
int Lcadep(int u,int v) {
    return lg[pos[u]] - lg[pos[u] ^ pos[v]];
}
```

比如说最大子段和就可以考虑分成几部分处理：

- 左边。
- 右边。
- 两边。

GSS1 - Can you answer these queries I - 洛谷

```cpp
#include <bits/stdc++.h>
using namespace std;

template <typename T>
void r1(T &x) {
    x = 0; int f(1); char c(getchar());
    for(; !isdigit(c); c = getchar()) if(c == '-') f = - 1;
    for(; isdigit(c); c = getchar()) x = (x * 10) + (c ^ 48);
    x *= f;
}

template <typename T, typename... Args>
void r1(T &x, Args&...arg) {
    r1(x), r1(arg...);
}

const int maxn = (1 << 16) + 5;
const int mxlg = 16;
const int maxm = mxlg + 5;
int lg[maxn << 1];
int Len, n, a[maxn], m;

struct Seg {
    #define ls (p << 1)
    #define rs (p << 1 | 1)
    #define mid ((l + r) >> 1)
    int pos[maxn << 1], f[maxm][maxn], s[maxm][maxn];
    void build(int p,int l,int r){
        if(l == r) return pos[l] = p, void();
        int z = lg[p] + 1, sum(0), pr;
//        printf("%d : %d %d\n", p, z, mid);
        f[z][mid] = s[z][mid] = sum = pr = a[mid];
        for(int i = mid - 1; i >= l; -- i) {
//            printf("i = %d\n", i);
            sum += a[i];
            f[z][i] = max(f[z][i + 1] + a[i], a[i]);
            s[z][i] = max(sum, s[z][i + 1]);
        }
        for(int i = mid - 1; i >= l; -- i) f[z][i] = max(f[z][i], f[z][i + 1]);
        f[z][mid + 1] = s[z][mid + 1] = sum = a[mid + 1];
        for(int i = mid + 2; i <= r; ++ i) {
            sum += a[i];
            f[z][i] = max(f[z][i - 1] + a[i], a[i]);
            s[z][i] = max(sum, s[z][i - 1]);
        }
        for(int i = mid + 2; i <= r; ++ i) f[z][i] = max(f[z][i], f[z][i - 1]);
        build(ls, l, mid), build(rs, mid + 1, r);
    }

    int Ask(int l,int r) {
```

```
        if(l == r) return a[l];
        int x = lg[pos[l]] - lg[pos[l] ^ pos[r]];
        return max({f[x][l], f[x][r], s[x][l] + s[x][r]});
    }

    #undef ls
    #undef rs
    #undef mid
}T;

signed main() {
    int i, j;
    r1(n);
    for(i = 1; i <= n; ++ i) r1(a[i]);
    for(Len = 2; Len < n; Len <<= 1);
    for(i = 2; i <= Len * 2; ++ i) lg[i] = lg[i >> 1] + 1;
    T.build(1, 1, Len);
    r1(m);
    for(int _ = 1; _ <= m; ++ _) {
        int l, r; r1(l, r);
        printf("%d\n", T.Ask(l, r));
    }
    return 0;
}
```