【题解】P4694 【PA 2013】 Raper

题目链接
双倍经验：CF802O April Fools' Problem (hard)
三倍经验：CF802N April Fools' Problem (medium)

线段树模拟费用流做法

看的这篇题解，但它讲的并不是很清晰。

显然，可以如下建模：
$s \to i$，流量 1，费用 a_i；
$i \to i+1$，流量 ∞，费用 0；
$i \to t$，流量 1，费用 b_i。
初始流量 k，求最小费用最大流。
直接跑 SPFA + EK 的最小费用最大流可以过掉 CF802N April Fools' Problem (medium)。
现在考虑用线段树模拟费用流。
根据 EK 算法，我们需要增广 k 次，每次必定是找到 $s \rightsquigarrow t$ 的最短路并增广之。
我们需要快速找到 $s \rightsquigarrow t$ 的最短路。

线段树节点信息

考虑维护什么信息，直接对着代码讲。

首先，为了方便，我们封装一下 `index` 类和 `path` 类，分别存储下标和路径，并重载小于运算符使得可以直接使用 `min` 函数合并（而不用写 `if`）。
显然最短路可以表示为 $s \to u \rightsquigarrow v \to t$，所以 `path` 只要存储 u, v 即可。
为方便，我们将 a_0, b_0 设为 ∞，后面要表示 ∞ 时直接将下标设为 0 即可。

```cpp
template<ll *arr>
struct index {
    int i;
    index() : i() {}
    index(int i) : i(i) {}
    inline ll operator () () const {
        return arr[i];
    }
    friend inline bool operator < (const index &A, const index &B) {
        return A() < B();
    }
};

struct path {
    index<a> u;
    index<b> v;
    path() : u(), v() {}
    path(index<a> u, index<b> v) : u(u), v(v) {}
    inline ll operator () () const {
        return u() + v();
    }
    friend inline bool operator < (const path &A, const path &B) {
        return A() < B();
    }
};
```

然后再来看节点信息。（命名奇怪请见谅）

```cpp
struct nde {
    /* record these things:
    pr: the shortest path from left to right in segment
    pl: the shortest path from right to left in segment
    pa: the shortest path available from right to left in segment
    wa: the index of minimum value of `a' in segment
    wb: the index of minimum value of `b' in segment
    ua: the index of minimum value of `a' reachable from left in
segment
    ub: the index of minimum value of `b' reachable from right in
segment
    flow: available flow from the segment's right point to left point
    tag: flow's segment add
    */
    path pr, pl, pa;
```

```
    index<a> wa, ua;
    index<b> wb, ub;
    ll flow, tag;
};
```

由于从左往右的路径容量均为 ∞，所以只用特别维护从右往左的路径的信息。
节点所代表的区间都是左闭右开的。具体的，会维护 $[l, r)$ 的所有 a, b 的值以及 $l \leftrightarrow l+1, l+1 \leftrightarrow l+2, \ldots, r-1 \leftrightarrow r$ 这些边。
维护 `pr` 表示节点所代表的区间内最短的从左往右的路径，`pl` 表示节点内最短的从右往左的路径，`pa` 表示节点内最短的**可行的**（也就是路径上剩余容量 > 0）从右往左的路径。其中 `pl` 是用来辅助计算 `pa` 的。
维护 `wa`，`wb` 表示节点所代表的区间内 a, b 的最小值的下标。维护 `ua` 表示区间内可以流到 l 处的最小的 a 的下标（也就是路径上剩余容量 > 0），`ub` 表示区间内可以从 r 处流到的最小的 b 的下标。
维护 `flow` 表示 $l \leftarrow l+1, l+1 \leftarrow l+2, \ldots, r-1 \leftarrow r$ 这些边的剩余容量的最小值，`tag` 是给这些边的剩余容量进行区间加法的标记。

在线段树建树时对叶子结点 $[l, l+1)$ 进行初始化，代码如下：

```
tr[x].pr = tr[x].pl = tr[x].pa = path(l, l);
tr[x].wa = index<a>(l);
tr[x].ua = index<a>(l);
tr[x].wb = index<b>(l);
tr[x].ub = index<b>(0);
tr[x].flow = 0;
tr[x].tag = 0;
```

因为区间是左闭右开的，维护的是 l 处的值和 $l \to l+1$ 这条边，故 `ua` 初始可行，为 l，`ub` 初始不可行，我们将下标设为 0 来将其设为 ∞。

pushup

代码如下：

```
friend inline nde operator + (const nde &A, const nde &B) {
    nde R;
    R.pr = min(min(A.pr, B.pr), path(A.wa, B.wb));
    R.pl = min(min(A.pl, B.pl), path(B.wa, A.wb));
```

```
    R.pa = min(min(A.pa, B.pa), path(B.ua, A.ub));

    R.wa = min(A.wa, B.wa);
    R.wb = min(A.wb, B.wb);

    R.ua = A.ua;
    R.ub = B.ub;

    R.flow = min(A.flow, B.flow);
    R.tag = 0;

    if (A.flow > B.flow) {
        Min(R.pa, A.pl);
        Min(R.pa, path(B.ua, A.wb));
        Min(R.ua, min(A.wa, B.ua));
    }
    else if (B.flow > A.flow) {
        Min(R.pa, B.pl);
        Min(R.pa, path(B.wa, A.ub));
        Min(R.ub, min(B.wb, A.ub));
    }

    return R;
}
```

pushup 需要特殊实现。

`if` 外面的部分相信都能看懂。下面详细讲解为什么会有这么奇怪的 `if`。
由于区间加法操作的存在，当左右子区间的 `flow` 相等时，即使它们大于 0 我们也不能将其视为可行的流。这是因为只要区间容量减 1，使得 `flow` 从 1 变成 0 时，区间的流变为不可行，但是线段树在处理时 不可能在遇到完整的区间时 继续向下分治 遍历区间内每个数来计算出现在的信息；而我们要想计算出 `ua`，`ub` 这些就只能通过遍历区间内每个数来计算出来。
因此我们考虑在左右子区间的 `flow` 不相同时将 `flow` 大的一边视为可行的，将小的一边视为不可行的；在左右子区间的 `flow` 相等时将整个区间视为不可行的。这样即使整个区间流量 -1，只要减的是整个区间，大的那一半区间 `flow` 永远是 >0 的。如果减的是大的那一半区间，自然会向下分治并 `pushup` 上来，信息依然是正确的。
这样做还有个细节，就是我们在线段树里同时维护 $n \to n+1$ 这条边并将其剩余容量视为 0。否则在 $1 \rightsquigarrow n$ 的所有边的剩余容量均 >0 时就没有更小的 `flow` 来与它比

较，就会导致 WA。下面是一个 hack：
input:

```
6 2
1 9 5 9 9 9
9 5 9 9 9 1
```

wrong output:

```
16
```

answer:

```
12
```

（有趣的是这个细节写错的代码在洛谷上能获得 96 分）

`if` 内的内容也很容易看懂吧，不讲了（逃）
大概就是当一个区间被视为可行的流时，我们将其 `pl` 也视为合法的 `pa`，将其 `wa`，`wb` 也视为合法的 `ua`，`ub`。

其他部分

`pushdown` 按照正常的来即可。
主函数部分也要特殊处理。在增广一条路径 $u \rightsquigarrow v$ 时，我们先进行区间加，再对 a_u, b_v 进行单点赋值设为 ∞。因为 `pushup` 建立在正确的 `flow` 上面，区间加是无法遍历到 u, v 的叶子节点及其祖先节点的，我们只能在单点赋值时对其进行 `pushup`，在此之前必须保证这些节点的 `flow` 是正确的。

Code

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

namespace MyNamespace {
```

```
typedef long long ll;

inline namespace MyIO {
    inline ll rd() {
        char ch = getchar();
        ll s = 0, w = 1;
        while (!isdigit(ch)) {
            if (ch == '-') w = -1;
            ch = getchar();
        }
        while (isdigit(ch)) {
            s = (s << 3) + (s << 1) + (ch ^ 48);
            ch = getchar();
        }
        return (s * w);
    }
    template<typename T>
    inline void wr(T x) {
        if (x < 0) x = -x, putchar('-');
        if (x > 9) wr(x / 10);
        putchar(x % 10 + 48);
    }
    inline void wrsp() {
        // pass
    }
    template<typename T, typename... Args>
    inline void wrsp(T x, Args... args) {
        wr(x), putchar(' '), wrsp(args...);
    }
    inline void wrln() {
        putchar('\n');
    }
    template<typename T>
    inline void wrln(T x) {
        wr(x), putchar('\n');
    }
    template<typename T, typename... Args>
    inline void wrln(T x, Args... args) {
        wr(x), putchar(' '), wrln(args...);
    }
} // namespace MyIO

inline namespace Base {
    #define bug(x) << #x << '=' << (x) << ' '
    #define siz(A) ll((A).size())

    template<typename T>
```

```cpp
    constexpr inline void Max(T &x, T y) {
        x = max(x, y);
    }
    template<typename T>
    constexpr inline void Min(T &x, T y) {
        x = min(x, y);
    }
} // namespace Base

constexpr ll INF = 1e18;

constexpr int fn = 5e5;
constexpr int N = fn + 10;

int n, kn;
ll a[N], b[N];

template<ll *arr>
struct index {
    int i;
    index() : i() {}
    index(int i) : i(i) {}
    inline ll operator () () const {
        return arr[i];
    }
    friend inline bool operator < (const index &A, const index &B) {
        return A() < B();
    }
};

struct path {
    index<a> u;
    index<b> v;
    path() : u(), v() {}
    path(index<a> u, index<b> v) : u(u), v(v) {}
    inline ll operator () () const {
        return u() + v();
    }
    friend inline bool operator < (const path &A, const path &B) {
        return A() < B();
    }
};

struct SGT {
    int n;
    struct nde {
        /* record these things:
```

```
        pr: the shortest path from left to right in segment
        pl: the shortest path from right to left in segment
        pa: the shortest path available from right to left in segment
        wa: the index of minimum value of `a' in segment
        wb: the index of minimum value of `b' in segment
        ua: the index of minimum value of `a' reachable from left in
segment
        ub: the index of minimum value of `b' reachable from right in
segment
        flow: available flow from the segment's right point to left
point
        tag: flow's segment add
        */
        path pr, pl, pa;
        index<a> wa, ua;
        index<b> wb, ub;
        ll flow, tag;
        friend inline nde operator + (const nde &A, const nde &B) {
            nde R;
            R.pr = min(min(A.pr, B.pr), path(A.wa, B.wb));
            R.pl = min(min(A.pl, B.pl), path(B.wa, A.wb));
            R.pa = min(min(A.pa, B.pa), path(B.ua, A.ub));

            R.wa = min(A.wa, B.wa);
            R.wb = min(A.wb, B.wb);

            R.ua = A.ua;
            R.ub = B.ub;

            R.flow = min(A.flow, B.flow);
            R.tag = 0;

            if (A.flow > B.flow) {
                Min(R.pa, A.pl);
                Min(R.pa, path(B.ua, A.wb));
                Min(R.ua, min(A.wa, B.ua));
            }
            else if (B.flow > A.flow) {
                Min(R.pa, B.pl);
                Min(R.pa, path(B.wa, A.ub));
                Min(R.ub, min(B.wb, A.ub));
            }

            return R;
        }
        inline nde& operator *= (ll t) {
            flow += t;
```

```
            tag += t;
            return *this;
        }
    } tr[N * 4];

    void constr(int _n) {
        n = _n + 1;
        bld(1, 1, n);
    }
    inline void pus(int x) {
        tr[x] = tr[x << 1] + tr[x << 1 | 1];
    }
    inline void dwn(int x) {
        if (tr[x].tag) {
            tr[x << 1] *= tr[x].tag;
            tr[x << 1 | 1] *= tr[x].tag;
            tr[x].tag = 0;
        }
    }

    void bld(int x, int l, int r) {
        // cerr bug(x) bug(l) bug(r) << endl;
        if (l + 1 == r) {
            tr[x].pr = tr[x].pl = tr[x].pa = path(l, l);
            tr[x].wa = index<a>(l);
            tr[x].ua = index<a>(l);
            tr[x].wb = index<b>(l);
            tr[x].ub = index<b>(0);
            tr[x].flow = 0;
            tr[x].tag = 0;
            return;
        }

        int o = ((l + r) >> 1);
        bld(x << 1, l, o);
        bld(x << 1 | 1, o, r);

        pus(x);
    }

    template<ll *arr>
    void upd_sgl_chg(int x, int l, int r, int u, ll k) {
        if (l + 1 == r) {
            arr[u] = k;
            return;
        }
```

```
        dwn(x);
        int o = ((l + r) >> 1);
        if (u < o)
            upd_sgl_chg<arr>(x << 1, l, o, u, k);
        else
            upd_sgl_chg<arr>(x << 1 | 1, o, r, u, k);

        pus(x);
    }
    template<ll *arr>
    inline void upd_sgl_chg(int u, ll k) {
        if (u <= 0 || u >= n) return;
        upd_sgl_chg<arr>(1, 1, n, u, k);
    }

    void upd_sgm_add(int x, int l, int r, int u, int v, ll k) {
        if (u <= l && r <= v) {
            tr[x] *= k;
            return;
        }

        dwn(x);
        int o = ((l + r) >> 1);
        if (u < o)
            upd_sgm_add(x << 1, l, o, u, v, k);
        if (v > o)
            upd_sgm_add(x << 1 | 1, o, r, u, v, k);

        pus(x);
    }
    inline void upd_sgm_add(int u, int v, ll k) {
        Max(u, 1), Min(v, n);
        if (u >= v) return;
        upd_sgm_add(1, 1, n, u, v, k);
    }

    nde qry_nde(int x, int l, int r, int u, int v) {
        if (u <= l && r <= v) {
            return tr[x];
        }

        dwn(x);
        int o = ((l + r) >> 1);
        if (v <= o) return qry_nde(x << 1, l, o, u, v);
        else if (u >= o) return qry_nde(x << 1 | 1, o, r, u, v);
        else return qry_nde(x << 1, l, o, u, o) + qry_nde(x << 1 | 1,
o, r, o, v);
```

```
    }
    inline nde qry_nde(int u, int v) {
        Max(u, 1), Min(v, n);
        return qry_nde(1, 1, n, u, v);
    }
} sgt;

ll ans;
void NamespaceMain() {
    n = rd(), kn = rd();
    for (int i = 1; i <= n; ++i)
        a[i] = rd();
    for (int i = 1; i <= n; ++i)
        b[i] = rd();
    a[0] = b[0] = INF;

    sgt.constr(n);
    for (int h = 1; h <= kn; ++h) {
        path p = min(sgt.tr[1].pr, sgt.tr[1].pa);
        // if (sgt.tr[1].flow > 0) Min(p, sgt.tr[1].pl);
        // cerr bug(p.u.i) bug(p.v.i) bug(sgt.tr[1].pa.u.i)
bug(sgt.tr[1].pa.v.i)
        // bug(sgt.qry_nde(4, 7).pl.u.i) bug(sgt.qry_nde(4, 7).pl.v.i)
        // bug(sgt.qry_nde(5, 7).ua.i) bug(sgt.qry_nde(4, 5).ub.i) <<
endl;
        if (p() >= INF) {
            #warning assert
            assert(0);
        }
        ans += p();
        if (p.u.i < p.v.i) sgt.upd_sgm_add(p.u.i, p.v.i, 1);
        else if (p.u.i > p.v.i) sgt.upd_sgm_add(p.v.i, p.u.i, -1);
        sgt.upd_sgl_chg<a>(p.u.i, INF);
        sgt.upd_sgl_chg<b>(p.v.i, INF);
    }

    wr(ans);
}
}
int main() {
    MyNamespace::NamespaceMain();
    return 0;
}
```

wqs 二分+反悔贪心做法

见这篇题解，写的很清楚了。
很唐的是 CF 数据，wqs 二分多点共线的情况，把选的数量 写成二分函数中尽可能多的选取的个数 而不是题目给定的 k，能过。洛谷的 hack 数据能卡掉。

Code

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

namespace MyNamespace {
typedef long long ll;

inline namespace MyIO {
    inline ll rd() {
        char ch = getchar();
        ll s = 0, w = 1;
        while (!isdigit(ch)) {
            if (ch == '-') w = -1;
            ch = getchar();
        }
        while (isdigit(ch)) {
            s = (s << 3) + (s << 1) + (ch ^ 48);
            ch = getchar();
        }
        return (s * w);
    }
    template<typename T>
    inline void wr(T x) {
        if (x < 0) x = -x, putchar('-');
        if (x > 9) wr(x / 10);
        putchar(x % 10 + 48);
    }
    inline void wrsp() {
        // pass
    }
    template<typename T, typename... Args>
    inline void wrsp(T x, Args... args) {
        wr(x), putchar(' '), wrsp(args...);
    }
    inline void wrln() {
        putchar('\n');
    }
    template<typename T>
    inline void wrln(T x) {
        wr(x), putchar('\n');
```

```
    }
    template<typename T, typename... Args>
    inline void wrln(T x, Args... args) {
        wr(x), putchar(' '), wrln(args...);
    }
} // namespace MyIO

inline namespace Base {
    #define bug(x) << #x << '=' << (x) << ' '
    #define siz(A) ll((A).size())

    template<typename T>
    constexpr inline void Max(T &x, T y) {
        x = max(x, y);
    }
    template<typename T>
    constexpr inline void Min(T &x, T y) {
        x = min(x, y);
    }
} // namespace Base

constexpr int fn = 1e6;
constexpr int N = fn + 10;

int n, kn, a[N], b[N];

struct nde {
    ll val; int cnt;
    friend inline bool operator < (const nde &A, const nde &B) {
        return A.val > B.val;
    }
};
inline nde solve(ll C) {
    priority_queue<nde> que;
    ll sum = 0; int cnt = 0;
    for (int i = 1; i <= n; ++i) {
        que.push({a[i] - C, 1});
        if (!que.empty() && b[i] + que.top().val <= 0) { // choose as
most as possible
            sum += b[i] + que.top().val;
            cnt += que.top().cnt;
            que.pop();
            que.push({-b[i], 0});
        }
    }
    return {sum, cnt};
}
```

```cpp
void NamespaceMain() {
    n = rd(), kn = rd();
    for (int i = 1; i <= n; ++i)
        a[i] = rd();
    for (int i = 1; i <= n; ++i)
        b[i] = rd();

    ll l = 0, r = 2e9 + 1; // [)
    while (l < r) {
        ll mid = ((l + r) >> 1);
        if (solve(mid).cnt >= kn)
            r = mid;
        else
            l = mid + 1;
    }

    nde pi = solve(r);
    // cerr bug(r) bug(pi.val) bug(pi.cnt) << endl;
    wr(pi.val + kn * r);
}
}
int main() {
    MyNamespace::NamespaceMain();
    return 0;
}
```