其他

分数规划

给出 a_i 和 b_i, 求一组 $w_i \in \{0,1\}$, 最小化或最大化

$$\frac{\sum\limits_{i=1}^{n} a_i \times w_i}{\sum\limits_{i=1}^{n} b_i \times w_i}$$

二分 mid:

$$\begin{aligned}
&\frac{\sum a_i \times w_i}{\sum b_i \times w_i} > mid \\
\Longrightarrow &\sum a_i \times w_i - mid \times \sum b_i \cdot w_i > 0 \\
\Longrightarrow &\sum w_i \times (a_i - mid \times b_i) > 0
\end{aligned}$$

```
int a[maxn], b[maxn];
double c[maxn];
bool check(double m)
{
    for(int i=1;i<=n;i++) c[i] = a[i]-m*b[i];
    sort(c+1, c+1+n, greater<double>());
    return accumulate(c+1, c+1+k, 0.0)>0;
}
double l=0, r=1e5;
while(r-l>eps)
{
    double mid = (l+r)/2;
    if(check(mid)) l = mid;
    else r = mid;
}
```

例: 分母至少为 W

```
double dp[maxw];
bool check(double m)
{
    fill(dp+1, dp+1+W, -1e9);
    for(int i=1;i<=n;i++)
    {
        for(int j=W;j>=0;j--)
        {
            int k = min(W, j+b[i]);
            dp[k] = max(dp[k], dp[j]+a[i]-m*b[i]);
        }
    }
    return dp[W]>0;
}
```

extc++(pd_ds & rope)

哈希表

用法同 `std::unordered_map`

```
__gnu_pbds::cc_hash_table<K, V> h; // 拉链法
__gnu_pbds::gp_hash_table<K, V> h; // 探测法
```

平衡树

```
__gnu_pbds::tree<pii, __gnu_pbds::null_type, less<pii>,
    __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update> rbt;
// $1: key type
// $2: val type(allow null_type)
// $3: comp
// $4: which tree (rbt, splay, ov)
// $5: node updater
```

```
rbt.insert(make_pair(x,i)); // insert, use pair to unique(let i>0 and unique)
rbt.erase(rbt.lower_bound(make_pair(x,0))); // remove
rbt.order_of_key(make_pair(x,0))+1; // query order(1-index) by number
rbt.find_by_order(x-1)->first; // query number by order(1-index)
rbt.find_by_order(rbt.order_of_key(make_pair(x,0))-1)->first; // query prev
rbt.find_by_order(rbt.order_of_key(make_pair(x+1,0)))->first; // query next

rbt.join(t); // merge t into rbt
rbt.split(x,t); // split elements greater than x to t;
```

Trie

```
__gnu_pbds::trie<string, null_type, __gnu_pbds::trie_string_access_traits<>,
    __gnu_pbds::pat_trie_tag, __gnu_pbds::trie_prefix_search_node_update> t;
// $1: key type
// $2: val type(allow null_type)
// $3: access_trait
// $4: recommand pat
// $5: node updater
```

堆

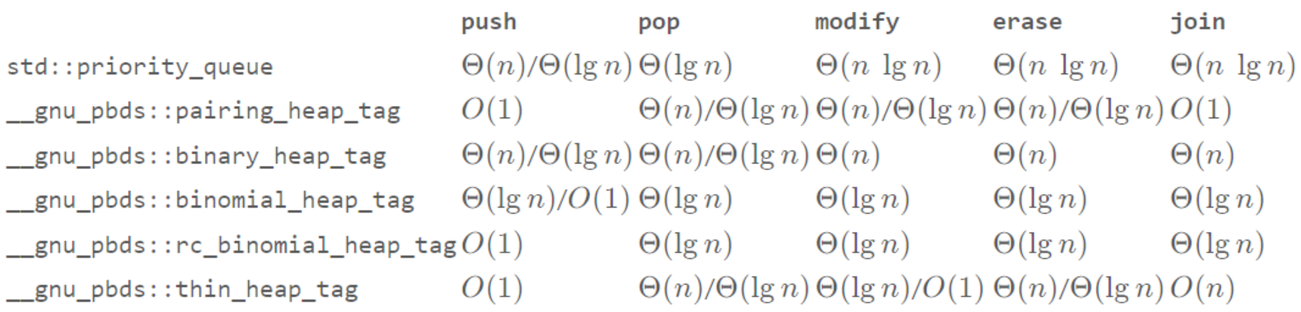
```
__gnu_pbds::priority_queue<int, less<>, __gnu_pbds::pairing_heap_tag> pq;
// $1: val type
// $2: less: 大根堆
// $3: 见下图
```

rope

比较暴力的长 `std::string`

- `operator+()` 与 `operator+=()`, 拼接
- `operator-()` 与 `operator-=()`, 剪切
- `operator<()` 与 `operator==()`, 比较

`rope` 暴力可持久化数组



	push	pop	modify	erase	join
std::priority_queue	$\Theta(n)/\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(n\ \lg n)$	$\Theta(n\ \lg n)$	$\Theta(n\ \lg n)$
__gnu_pbds::pairing_heap_tag	$O(1)$	$\Theta(n)/\Theta(\lg n)$	$\Theta(n)/\Theta(\lg n)$	$\Theta(n)/\Theta(\lg n)$	$O(1)$
__gnu_pbds::binary_heap_tag	$\Theta(n)/\Theta(\lg n)$	$\Theta(n)/\Theta(\lg n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
__gnu_pbds::binomial_heap_tag	$\Theta(\lg n)/O(1)$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$
__gnu_pbds::rc_binomial_heap_tag	$O(1)$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$
__gnu_pbds::thin_heap_tag	$O(1)$	$\Theta(n)/\Theta(\lg n)$	$\Theta(\lg n)/O(1)$	$\Theta(n)/\Theta(\lg n)$	$O(n)$

Figure 1: pb_ds_heap

```
int n,m;
cin>>n>>m;
vector<__gnu_cxx::rope<int>> w(1);
w.front().push_back(0);
for(int i=1;i<=n;i++)
{
    int x;
    cin>>x;
    w.front().push_back(x);
}
for(int v=1;v<=m;v++)
{
    int r,o,p;
    cin>>r>>o>>p;
    w.emplace_back(w[r]);
    if(o==1)
    {
        int x;
        cin>>x;
        w[v].mutable_reference_at(p) = x;
    }
    else cout<<w[v][p]<<'\n';
}
```

rope 暴力文艺平衡树

```
__gnu_cxx::rope<int> a,b;
int n,m;
cin>>n>>m;
for(int i=1;i<=n;i++) a.push_back(i), b.push_back(n-i+1);
while(m--)
{
    int l,r;
    cin>>l>>r;
    l--;
    auto p = a.substr(a.begin()+l, a.begin()+r);
    a = a.substr(a.begin(), a.begin()+l)+b.substr(b.begin()+(n-r), b.begin()+(n-l))+ \
        a.substr(a.begin()+r, a.end());
    b = b.substr(b.begin(), b.begin()+(n-r))+p+b.substr(b.begin()+(n-l), b.end());
}
for(auto i : a) cout<<i<<' ';
cout<<endl;
```