贪心算法

51Nod wwwwodddd

前言

- 感谢51Nod给我此次直播的机会。
- 51Nod要做国内最好的程序员社区。
- 51Nod的5000人QQ群251587768。

A. 低买高卖

- 考虑股票市场，已知n天的价钱。
- 每天可以买入1，或者卖出1，或者什么都不做。
- 问n天之后剩下最多多少钱?

做法讨论

- 每到新的一天，直接把当天的股票卖掉。
- 但是因为现在没有股票，所以我们需要把今天或之前的某次卖改成持有，或者把持有，改成买。
- 用一个堆实现就可以了。

参考代码

- 读入一天的价钱x。
- 可以把卖出改为持有。
- 可以把持有改为买入。
- 入队2次，表示有2次修改机会
- 卖出收益x减去买入成本q.top()
- q是大根堆，根是价钱最小的。
- 卖出必须在买入之前。

```cpp
#include <bits/stdc++.h>
using namespace std;
priority_queue<int> q;
int n, x;
long long z;
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &x);
        q.push(-x);
        z += x + q.top();
        q.pop();
    }
    printf("%lld\n", z);
    return 0;
}

```

B. 排队接水

- n个人一起接水，第i个人需要b[i]的时间来接水。
- 最小化所有人等待时间的总和。

做法讨论一

- 倒数第i个人，带来的贡献是i * b。
- b比较大的，放在后面，b比较小的，放在前面

做法讨论二

- 考虑相邻2个人，假设他们的时间是b[x], b[y]
- 如果x在前面，代价是2*b[x]+b[y]
- 如果y在前面，代价是2*b[y]+b[x]
- 两者比较，小的在前面。

参考代码

- 直接sort排序，小在前。
- a[i]的系数是n-i。
- 统计答案。

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, a[1020];
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
    sort(a, a + n);
    int s = 0;
    for (int i = 0; i < n; i++) {
        s += a[i] * (n - i);
    }
    printf("%d\n", s);
    return 0;
}

```

C. 排队接水二

- n个人一起接水，第i个人需要b[i]的时间来接水，重要性是a[i]。
- 最小化所有人等待时间乘以自己的重要性的总和。

做法讨论二

- 仿照上个题目，考虑相邻2个人x, y。
- 如果x在前面，代价是a[x]*b[x]+a[y]*(b[x]+b[y])
- 如果y在前面，代价是a[y]*b[y]+a[x]*(b[y]+b[x])
- 如果b[x]*a[y]<b[y]*a[x]，x在前，否则y在前。
- 几乎等价于按b[i]/a[i]排序，小在前。

一个小坑

- 按b[i]/a[i]排序，需要处理a[i]为0的情况。
- 转化为乘法，不需要考虑a[i]为0的情况。
- 但是如果a[i]=0且b[i]=0。
- 比较函数会认为任意分数等于0/0，导致sort出错

参考代码

- 使用pair
- sort比较函数，按比例排序
- 忽略a[i]或b[i]为0的
- 最后统计答案

```cpp
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cassert>
using namespace std;
pair<int, int>a[1000020];
int n;
bool cmp(const pair<int, int> &a, const pair<int, int> &b) {
    return a.first * b.second < b.first * a.second;
}
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &a[i].first, &a[i].second);
        if (a[i].first == 0 || a[i].second == 0) {
            i--;
            n--;
        }
    }
    sort(a, a + n, cmp);
    long long z = 0, s = 0;
    for (int i = 0; i < n; i++) {
        s += a[i].first;
        z += s * a[i].second;
    }
    printf("%lld\n", z);
    return 0;
}
```

D. 做任务一

- 输入n个任务的起点s[i]和终点e[i]。
- 问一个人最多能做多少任务。

做法讨论一

- 按右端点排序。
- 从左向右扫，如果当前能做，那么必须做这个。
- 因为当前扫到的是结束时间最早的任务。

参考代码一

- 读入先a[i].second，再a[i].first
- 排序
- 维护一个r
- 能做，就做；做不了，放弃。

```cpp
#include <bits/stdc++.h>
using namespace std;
int t, n, m;
pair<int, int> a[100020];
int main() {
    scanf("%d", &t);
    for (int tt = 0; tt < t; tt++) {
        int z = 0;
        multiset<int> s;
        scanf("%d%d", &n, &m);
        for (int i = 0; i < n; i++) {
            scanf("%d%d", &a[i].second, &a[i].first);
        }
        sort(a, a + n);
        int r = 0;
        for (int i = 0; i < n; i++) {
            if (r <= a[i].second) {
                r = a[i].first;
                z++;
            }
        }
        printf("%d\n", z);
    }
    return 0;
}
```

做法讨论二

- 按左端点排序。
- 从左向右扫，如果当前能做，那么试图做这个。
- 如果做不了，并且这个任务的结束时间比当前的结束时间早，那么放弃原本的任务，转而做这个
- 因为按左端点排序，如此转换必定成立。

参考代码二

- 直接排序pair，默认先排first，再排second（先排左端点）
- 维护一个右端点r。
- 遇到任务，如果可以做，就做
- 如果不可以做，试图换。

```cpp
#include <bits/stdc++.h>
using namespace std;
int t, n, m;
pair<int, int> a[100020];
int main() {
    scanf("%d", &t);
    for (int tt = 0; tt < t; tt++) {
        int z = 0;
        multiset<int> s;
        scanf("%d%d", &n, &m);
        for (int i = 0; i < n; i++) {
            scanf("%d%d", &a[i].first, &a[i].second);
        }
        sort(a, a + n);
        int r = 0;
        for (int i = 0; i < n; i++) {
            if (r <= a[i].first) {
                r = a[i].second;
                z++;
            } else if (a[i].second < r) {
                r = a[i].second;
            }
        }
        printf("%d\n", z);
    }
    return 0;
}
```

E. 做任务三

- 输入n个任务的起点s[i]和终点e[i]。
- 问最少需要几个人做完这些任务。

做法讨论二

- 按左端点排序。
- 从左向右扫，如果当前能做，那么试图做这个。
- 所谓能做，是指当前最早闲下来的人，可以做。
- 如果做不了，新开一个人，做这个任务。
- 需要用priority queue实现。

参考代码二

- 队列维护所有人的结束时间。
- priority queue默认大根堆，选择加入相反数，实现小根堆
- q.top()最小的即结束最早。
- 遇到一个新的任务，试图让q.top()来接，如果能接，更新
- 如果不能接，多开一个人，更新

```cpp
#include <bits/stdc++.h>
using namespace std;
int t, n, m;
pair<int, int> a[100020];
int main() {
    scanf("%d", &t);
    for (int tt = 0; tt < t; tt++) {
        int z = 0;
        priority_queue<int> q;
        scanf("%d%d", &n, &m);
        for (int i = 0; i < n; i++) {
            scanf("%d%d", &a[i].first, &a[i].second);
        }
        sort(a, a + n);
        for (int i = 0; i < n; i++) {
            if (q.size() && -q.top() <= a[i].first) {
                q.pop();
                q.push(-a[i].second);
            } else {
                q.push(-a[i].second);
                z++;
            }
        }
        printf("%d\n", z);
    }
    return 0;
}
```

做法讨论一

- 按右端点排序，依次处理所有任务
- 对于每个任务，选择最晚闲下来人做这个任务。
- 如果没有人能做，新开一个人做这个任务。
- 最晚闲下来的人，需要用set的upper bound实现

参考代码二

- multiset维护所有人的结束时间
- 如果能接，找到<=此任务开始时间，最晚的人来接。
- 先upper bound找到>此任务开始时间，最早的，然后自减。
- 如果接不了，直接新找一个人

```cpp
#include <bits/stdc++.h>
using namespace std;
int t, n, m;
pair<int, int> a[100020];
int main() {
    scanf("%d", &t);
    for (int tt = 0; tt < t; tt++) {
        int z = 0;
        multiset<int> s;
        scanf("%d%d", &n, &m);
        for (int i = 0; i < n; i++) {
            scanf("%d%d", &a[i].second, &a[i].first);
        }
        sort(a, a + n);
        for (int i = 0; i < n; i++) {
            if (s.size() && *s.begin() <= a[i].second) {
                s.erase(--s.upper_bound(a[i].second));
                s.insert(a[i].first);
            } else {
                z++;
                s.insert(a[i].first);
            }
        }
        printf("%d\n", z);
    }
    return 0;
}
```

F. 字符串连接

- 输入n个字符串s[i]
- 将他们按某个顺序连接起来
- 使得字典序最小

做法讨论

- 最容易想到的是按字典序排序。
- 一个反例是ba b，答案是bab而不是bba。
- 空字符最大还是最小？（字典中是最小）
- 如果认为是最大的话，反例是bc b，答案是bbc

做法讨论

- 对于任意2个字符串，如果交换后更优，就交换。
- 类似冒泡排序，相当于按照a + b < b + a排序。
- 可以直接sort，自定义比较函数。
- 这个题在NOIP 1998就出过。

参考代码

- 读入n，读入n个字符串。
- 比较函数直接返回a+b<b+a
- 输出n个字符串即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
bool cmp(const string &a, string &b) {
    return a + b < b + a;
}
int n;
string s[120];
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> s[i];
    }
    sort(s, s + n, cmp);
    for (int i = 0; i < n; i++) {
        cout << s[i];
    }
    cout << endl;
    return 0;
}
```

G. 缓存交换

- 输入缓存的容量m，和需要访问的n个内存位置。
- 问最少需要几次，将内存内容装入缓存。

背景介绍

- 这是一个经典问题。
- 在实际应用中，并不能知道自己未来要访问哪里
- 所以有各种各样的近似算法，举2个例子。
- FIFO，相当于队列，把最先加入的替换掉。
- LRU，把最久没有用过的换掉。
- https://en.wikipedia.org/wiki/Cache_replacement_policies

做法讨论

- 本题已知未来的访问位置，所以有理论最优解。
- 应该把下一次出现的位置，最远的换掉。

参考代码

- p[i]表示下一次a[i]出现的位置
- 如果没有下一次，那么设为n+1
- 每次找一下是否出现了。
- 没出现：删掉最晚的，加入下一次答案++
- 出现了：删掉本次，加入下一次，答案不变。

```cpp
#include <bits/stdc++.h>
using namespace std;
multiset<int> s;
map<int, int>g;
int p[100020];
int a[100020];
int n, m, z;
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
    }
    for (int i = 1; i <= n; i++) {
        p[i] = n + 1;
        p[g[a[i]]] = i;
        g[a[i]] = i;
    }
    for (int i = 1; i <= n; i++) {
        if (s.find(i) != s.end()) {
            s.erase(i);
            s.insert(p[i]);
        } else {
            z++;
            if (s.size() == m) {
                s.erase(--s.end());
            }
            s.insert(p[i]);
        }
    }
    printf("%d\n",z);
    return 0;
}
```

H. 挑剔的美食家

- n个奶牛，每个奶牛想吃草。
- 第i个奶牛的草，要求价钱>=a[i]，品质>=b[i]。
- m个草，每个草只能被吃一次。
- 第i个草，价钱c[i]，品质d[i]。
- 为每个奶牛准备一份草，最小化钱数。

做法讨论

- 将奶牛和草，按照品质从大到小排序，
- 从大到小依次处理所有奶牛。对于每个奶牛，把符合自己要去的所有草的价钱，都加入multiset。
- 取出符合条件的，最便宜的草。
- 如果没有符合条件的草，那么无解。

参考代码

- 排序所有时间，按照品质从大到小。（取相反数，从小到大
- 对于每个牛，先加入所有符合条件的草的价钱。
- 取出其中>=要求且最便宜的，用lower bound来做。
- 如果不存在，那么无解
- 如果存在，加入答案

```cpp
#include <bits/stdc++.h>
using namespace std;
pair<int, int> a[100020];
pair<int, int> b[100020];
int n, m, p;
long long z;
multiset<int>s;
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) {
        scanf("%d%d", &a[i].second, &a[i].first);
        a[i].first = -a[i].first;
    }
    for (int i = 0; i < m; i++) {
        scanf("%d%d", &b[i].second, &b[i].first);
        b[i].first = -b[i].first;
    }
    sort(a, a + n);
    sort(b, b + m);
    for (int i = 0; i < n; i++) {
        while (p < m && b[p].first <= a[i].first) {
            s.insert(b[p++].second);
        }
        multiset<int>::iterator it = s.lower_bound(a[i].second);
        if (it == s.end()) {
            z = -1;
            break;
        } else {
            z += *it;
            s.erase(it);
        }
    }
    printf("%lld\n", z);
    return 0;
}
```

I. 最高的奖励

- 输入n个任务，每个任务有时刻E[i]和收益W[i]。
- 每个任务需要花费1的时间，
  如果在E[i]之前做完，那么有W[i]，否则没有收益。
- 初始时间是0，问最终最大收益是多少?
- 2 <= n <= 50000
- 1 <= E[i] <= 1e9, 1 <= W[i] <= 1e9

做法讨论

- 将所有任务按时刻从小到大排序，
  如果2个任务时刻一样，那么他们任意顺序都可以。

- 维护一个优先队列（堆）

- 按照时间顺序从前到后处理每个任务，
  先将W[i]加入队列，如果此时队列大小大于E[i]
  删除队列中W[i]最小的。
  即在任意时刻E[i]之前，有至多E[i]个任务需要完成。

- 时间复杂度O(n log n)。

参考代码

- 使用pair存储时间。
- 直接排序，first是时刻。
- 使用优先队列，
  （或multiset，不要用set）
- 注意优先队列是大根堆。
  要加入相反数，
  退出是更新答案注意符号。
- 输出答案。

```cpp
#include <iostream>
#include <algorithm>
#include <queue>
using namespace std;

pair<int, int> a[50000];
priority_queue<int>q;
int n;
long long ans;
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i].first >> a[i].second;
    }
    sort(a, a + n);
    for (int i = 0; i < n; i++) {
        q.push(-a[i].second);
        ans += a[i].second;
        if (q.size() > a[i].first) {
            ans += q.top();
            q.pop();
        }
    }
    cout << ans << endl;
    return 0;
}
```

J. 夹克老爷的逢三抽一

- n = 3k个数m[i]排成一个环形。
- 每次选取1个数m[i]，并且删掉这个数m[i]和他两边的数m[i-1], m[i+1]，不能再选。
- 一共选k次，最大化选出的k个数字的和。
- n < 100000, 0 <= m[i] <= 1e9

做法讨论

- ~~回顾bzoj1150 [CTSC2007]数据备份Backup~~
- 每次选出最大的一个数字m[i]。
- 将m[i-1], m[i], m[i+1]删去，
  并加入m[i-1]+m[i+1]-m[i]。
- 上述过程重复k次。时间复杂度O(n log n)。

参考代码

- 使用双向链表和set
  set记录下标，不需要multiset

- 读入，构造双向链表和set

- 循环k次每次选取出最大的下标j
  最大的左边的数a，最大的b
  最大的右边的数c

- 删除j的左右两个数，
  修改m[j] = a + b - c
  更新j（先erase再insert）

```cpp
#include <iostream>
#include <set>
using namespace std;
int L[99999], R[99999];
long long m[99999];
set<pair<long long, int> >s;
void insert(int x) {
    s.insert(make_pair(m[x], x));
}
void erase(int x) {
    s.erase(make_pair(m[x], x));
}
void del(int x) {
    erase(x);
    R[L[x]] = R[x];
    L[R[x]] = L[x];
}
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> m[i];
        insert(i);
        L[(i + 1) % n] = i;
        R[i] = (i + 1) % n;
    }
    long long ans = 0;
    for (int i = 0; i < n / 3; i++) {
        int j = s.rbegin()->second;
        long long a = m[L[j]];
        long long b = m[j];
        long long c = m[R[j]];
        ans += b;
        del(L[j]);
        del(R[j]);
        erase(j);
        m[j] = a + c - b;
        insert(j);
    }
    cout << ans << endl;
    return 0;
}
```

更多的说明

- 将a, b, c换成a+c-b可以理解为，选b之后后悔，改成a和c，这样增加的收益是a+c-b。
- 因此我们可以根据最终结果反推出，在长度为n的环中选k=n/3个数，且这些数互不相邻的方案。
- 但是可能有1 0 2 0 3 0 4 0 5这种情况，我们选取了中间的3，2和4就相邻了，不能同时选择。
  下面我们来说明一定可以选出这k个数。
- 一定存在一个数，他的一边是两个未被选择的数，选取他。
  这样转化成了一个长度为n-3的环选k-1个数，数学归纳法即可

总结

- 总体来说贪心题都是比较简单的。
- 在NOIP中会以第一题，或者第二题出现。
- 常常和其他算法配合使用，比如堆，平衡树。
- 也有比较难的贪心题，但是那样的话我们一般就不认为是贪心了。而是数据结构，或者其他什么

谢谢大家

- 感谢51nod和夹克老爷的支持。