计试 2401 程设期中 非官网题解

BY 黄万平；Codeforces ID: hwpvector

前言

在没有认真地思考、尽自己最大的努力做完之前，请不要看题解。每一题最好都自己实现一遍，看题解明白思路不等于能实现出来。

每一题我都会将我的码给贴出。由于我的码风，以及习惯性的用大量 STL 和 Lambda 表达式，所以初学者可能看不太懂。

也欢迎大家继续编辑该题解（会附上 Markdown 格式的版本）。

Problem 1

题意：

> 给若干电阻值，请你计算出它们并联的电阻值

$$\frac{1}{R}=\sum_{i=1}^{n}\frac{1}{R_i}$$

按照公式枚举即可。

复杂度 $\Theta(n)$。

```
int main() {
    int n; cin>>n;
    vector<double> a(n);
    for (int i=0;i<n;i++) cin>>a[i];
    double ans=0;
    for (int i=0;i<n;i++) ans+=1/a[i];
    printf("%.4lf\n",1/ans);
    return 0;
}
```

Problem 2

题意：

> B 进制下的高精加

先分析，再写代码。Think Twice, Write Once

这个问题，和普通的高精加的区别，就是有了 B 进制这个条件。我们考虑怎样去搞一搞这个条件。

具体地，对于两个 B 进制下的一位数相加，我们考虑该怎么做。我们可以先写一个函数 `int toint (char c)`，将 B 进制下的字符 c 转换成其所对应的整数。然后整数相加。判断是否进位，如果进位了，这一位的数需要减去 B。我们最终输出的也是字符格式的 B 进制数，我们可以再写一个函数 `char tochar (int c)`，将整数 c 转换成其所对应的字符。

于是，接下来，所有的操作和朴素的高静加一模一样。具体的：

假设给定的 B 进制下的两个数为：

$$s = s_0 s_1 \cdots s_{n-1}$$
$$s' = s'_0 s'_1 \cdots s'_{m-1}$$

我们发现这两个数位数不同，而高精度加需要对两个字符串同时从后往前进行开始枚举。

我们可以这样子简化代码过程：先将 s, s' 字符串取反（让 $s_i = s_{n-1-i}, s'_i = s'_{n-1-i}$），然后对于每次相加的两个位置，下标都是相同的。最后再将结果取反即可。记：

$$s^R = s_{n-1} s_{n-2} \cdots s_0$$
$$s'^R = s'_{m-1} s'_{m-2} \cdots s'_0$$

并且，将 s^R 和 s'^R 中较短串，末尾不断补 0，使两串长度相同（相当于对于一开始的数补前导 0）。然后从前往后，枚举每一位。

在枚举的过程中，记录前一位是否进位。如果进位了，这一位需要额外加 1。一开始这个标记为否。

最后一位枚举完时，如果该进位标记为真，则需要再补一位。

若不管 B 的大小，复杂度为 $\Theta(\max\{n, m\})$

此题比较难实现。放一个我的实现，和上述题解过程略有出路（将补 0 的过程换了一种写法），可以看看不同的思路：

```
int main() {
    int b; cin>>b;
```

```
    string s1,s2;
    cin>>s1>>s2;
    reverse(s1.begin(),s1.end());
    reverse(s2.begin(),s2.end());
    auto toint=[](char c) {
        if (c>='0'&&c<='9') return c-'0';
        else return c-'A'+10;
    };
    auto tochar=[](int c) {
        if (c<=9) return (char)(c+'0');
        else return (char)(c-10+'A');
    };
    auto get1=[=](int i) {
        if (i<s1.size()) return s1[i];
        else return '0';
    };
    auto get2=[=](int i) {
        if (i<s2.size()) return s2[i];
        else return '0';
    };
    string ans;
    for (int i=0,add=0;i<max(s1.size(),s2.size())+1;i++) {
        int p=add+toint(get1(i))+toint(get2(i));
        if (p>=b) p-=b,add=1;
        else add=0;
        ans.push_back(tochar(p));
    }
    while (ans.back()=='0'&&ans.size()!=1) ans.pop_back();
    reverse(ans.begin(),ans.end());
    cout<<ans<<'\n'; return 0;
}
```

Problem 3

题意：

> 给一个 $n \times m$ 的矩阵 $a_{i,j}$。若 $a_{i,j}$ 不为 0，代表这个点有权重 $a_{i,j}$，保证权重各不相同。一人贪心地从第 0 行的任意位置开始，向当前存在权重最大的点走，获得该点的权重，并最终回到第 0 行的任意位置。每一步可以走相邻的点，代价为 1；获得该点的权重代价为 1。求在代价 K 内，最多能获得多少权重。

题意比较难懂。需要多读几遍。

先分析。有权重点的大小是固定的，访问它们的先后顺序也是固定的。所以，我们把所有有权重的点拿出来，记录它的 权重、横坐标、纵坐标，然后对权重从大到小排序，就是这些点的先后访问顺序。

考虑当前需要去访问的点为 now_x, now_y，它从前一个点 pre_x, pre_y 走过来需要的最小代价为 $|pre_x - now_x| + |pre_y - now_y|$，获得权重需要 1 的代价。

假设此时此刻，立刻往 0 行走，所需要的最小代价为 now_x（实际上可能并没有这么走，我们假设先这样走，如果这样走可行，我们再考虑从该点继续访问下一个点）。将前面的所有代价加起来，如果大于 K，则说明这个点访问不到，在前一个点访问后就向 0 行走了。为什么？如果继续访问其他点，那肯定会绕原路，代价肯定会更大。

但还有一个问题，就是在这个过程中涉及前一个点。那我们怎样去处理第一个点？读者自行思考。

复杂度为 $\Theta(n^2 \log n)$，瓶颈在排序。

```
int main() {
    int n,m,T; cin>>n>>m>>T;
    vector<array<int,3>> v;
    using ar3=array<int,3>;
    for (int i=1;i<=n;i++) {
        for (int j=1;j<=m;j++) {
            int x; cin>>x;
            if (x!=0) v.push_back({x,i,j});
        }
    }
    sort(v.begin(),v.end(),[](ar3 x,ar3 y){return x[0]>y[0];});
    // for (auto [x,y,z]:v) {cout<<x<<' '<<y<<' '<<z<<'\n';}
    if (v.size()==0) {cout<<0<<'\n';return 0;}
    ar3 st=*v.begin();
    int ans=st[0],nx=st[1],ny=st[2],t=nx+1;
    if (t+st[1]>T) {cout<<0<<'\n';return 0;}
    for (auto it=next(v.begin());it!=v.end();++it) {
        st=*it; t+=abs(nx-st[1])+abs(ny-st[2])+1;
        if (t+st[1]>T) {cout<<ans<<'\n';return 0;}
        nx=st[1],ny=st[2]; ans+=st[0];
    } cout<<ans<<'\n';
    return 0;
}
```

Problem 4

> 给定一个长度为 n 的数组 a_i，选出其中连续且非空的一段使得这段和最大

最朴素的做法，枚举每一段 $[l,r](1 \le l \le r \le n)$，暴力算出这一段之间的和，并对这些和取最大值。

容易验证，这样做的复杂度为 $O(n^3)$

如果我们考虑枚举 l，再枚举 r，每次和的增加量为 a_r——在已知 $\sum_{i=l}^{r-1} a_i$ 的和的情况下，可以 $O(1)$ 算出 $\sum_{i=l}^{r} a_i$ 的和。于是，我们得到了一个复杂度为 $O(n^2)$ 的做法。该做法显然能通过该题。

我们接下来拓展一些复杂度更优的做法。

前缀和 算法。能通过 $\Theta(n)$ 的预处理，$\Theta(1)$ 的算出一段区间的和。

我们记 $s_i = \sum_{j=1}^{i} a_j$。记 $s_0 = 0$

s 被称为前缀和数组。

我们可以通过转移：$s_{i+1} = s_i + a_{i+1}$，$\Theta(n)$ 的算出前缀和数组。

对于区间 $[l,r]$ 的和 $\sum_{i=l}^{r} a_i$，显然对于 $s_r - s_{l-1}$

于是，我们又得到了一个 $\Theta(n^2)$ 的做法：枚举每一段 $[l,r](1 \le l \le r \le n)$，利用前缀和 $O(1)$ 的算出每一段的和，并对这些和取最大值

我们肯定不满足于此。我们考虑我们所需要得到的最大值的式子：

$$s_r - s_{l-1}, (1 \le l \le r \le n)$$

容易把它变换成：

$$s_r - s_l, (0 \le l < r \le n)$$

我们考虑枚举上式的 r，对于每个 r，我们需要找到一个 $l, (0 \le l < r)$，使得 s_l 最小，此时式子最大。

我们再考虑维护一个前缀最小值数组 p_j，为 $s_i, i \in [0,j]$ 中的最小值。显然，$p_{j+1} = \min(p_j, a_{j+1})$，于是我们可以 $\Theta(n)$ 的求出 p_j。

$l, (0 \le l < r)$ 时，s_l 最小值显然等于 p_{r-1}

于是，我们得到了一个 $\Theta(n)$ 的算法。

```cpp
#define inf 0x3f3f3f3f
const int N=1e5+5;
int n,a[N],s[N],p[N];

int main() {
    cin>>n;
    s[0]=0;
    for (int i=1;i<=n;i++) {
        cin>>a[i];
        s[i]=s[i-1]+a[i];
    }
    p[0]=0;
    for (int i=1;i<=n;i++) {
        p[i]=min(p[i-1],s[i]);
    }
    int ans=-inf;
    for (int i=1;i<=n;i++) {
        ans=max(ans,s[i]-p[i-1]);
    }
    cout<<ans<<'\n';
    return 0;
}
```

Problem 5

> 找到数组中第 k 小的数（相同大小的数只计算一次）

就相当于找到集合中第 k 小的数。我们能够想到 STL 容器中的 `set` 恰能去重、实现这个功能。

```cpp
int main() {
    int n,k; cin>>n>>k; set<int> s;
    for (int i=1;i<=n;i++) {
        int x; cin>>x; s.insert(x);
    }
    if (s.size()<k) cout<<"NO RESULT\n";
    else {
        int cnt=0;
        for (auto x:s) {
            ++cnt;
            if (cnt==k) cout<<x<<'\n';
        }
```

```
    }
}
```

或者，对于排序数组，STL 中的 `unique` 也能实现去重。

上面两种做法的时间复杂度为 $\Theta(n \log n)$

当然，上述调用标准库的做法，可能比较不雅观。我简单说说不调用这些库的做法：

- 我们肯定要先进行排序，不然怎么求第 k 小，怎么知道那些数重复了？
- 我们考虑两种排序：桶排和快排
  - 桶排：我们将数据的值作为下标，存入数组中。从小到大访问这些数组的下标，如果这个桶中有数，直接将该数取出（不重复取出该桶中的所有元素，仅取一个）。然后查询第 k 小即可。复杂度 $\Theta(\max\{a_i\})$
  - 快排：直接对数组不去重地进行朴素的快速排序或者其他方式的排序。访问该排序数组，对于每个下标 i，如果 $i=1$ 或者 $a_i \neq a_{i-1}$，则说明 a_i 是首个不重复的数。然后查询第 k 小即可。复杂度为排序的复杂度，快排平均复杂度为 $\Theta(n \log n)$

Problem 6

该题笔者没有老老实实的用评测平台所述的算法写、给的代码补全。所以该题我仅放代码供大家参考，不再过多阐述。欢迎大家来完善该题的题解。

```cpp
int main() {
    vector<int> v; int n,m; cin>>n>>m;
    for (int i=1;i<=n+m;i++) {
        int x; cin>>x; v.push_back(x);
    }
    sort(v.begin(),v.end());
    int cnt=0;
    for (int i=0;i<v.size();i++) {
        if (i==0) {++cnt;continue;}
        if (v[i]!=v[i-1]) {
            if (cnt==1) cout<<v[i-1]<<' ';
            else cout<<v[i-1]<<'*'<<cnt<<' ';
            cnt=1;
        } else ++cnt;
    }
    if (cnt==1) cout<<v.back()<<'\n';
    else cout<<v.back()<<'*'<<cnt<<'\n';
    return 0;
}
```