第25次CSP解题报告

本次报告撰写人：张晨阳
题目：202203，第25次CSP

题目 1：未初始化警告

题目描述：

k 条赋值语句，n个变量：$a_1, a_2, \ldots, a_n$ ，常量a_0 。
第 i 条赋值语句为 $a_{x_i} = a_{y_i}$
输出多少条赋值语句右值未被初始化。
$0 < n, k \leq 10^5$

题解：

解法一：

用一个数组标记常量和变量。将已初始化的变量标记为1，未初始化标记为0。常量初始化为1.
注意：需要先判断右值是否初始化，再初始化左值。

时间复杂度$O(n)$，空间复杂度$O(n)$, 期望得分100.

3673266	张晨阳	张晨阳	未初始化警告	05-13 15:27	287B	CPP11	正确	100	140ms	3.402MB

C++

```cpp
#include<iostream>
# define N 100005
using namespace std;

int main()
{
	int k, n, lef, rig;
	cin >> n >> k;
	int x[N] = { 0 };
	x[0] = 1;
	int ans = 0;
	for (int i = 0; i < k; i++) {
		cin >> lef >> rig;
		if (!x[rig])
			ans++;
		x[lef] = 1;
	}
	cout << ans << endl;
	return 0;
}
```

题目 2：出行计划

题目描述：

t 时刻做核酸，$t+k$ 时刻得到结果。
若要求持 c 个单位时间结果，则 $t+k$ 到 $t+k+c-1$ 时刻得到的结果可用。
n 项出行计划，第 i 项为：t_i 时刻进入场所，该场所要求 c_i 个单位时间内的核酸结果。
m 个查询：在 q_m 时刻做了核酸，可满足多少项出行计划。
$0 < n, m, k \leq 10^5$，$0 < t_i, c_i, q_m \leq 2 \times 10^5$

题解：

解法一：

用一长度为$2N$的数组存储每个出行计划可进入的核酸时间，每个查询依次与每个计划的最早时间和最晚时间对比。

时间复杂度$O(mn)$，空间复杂度$O(n)$，期望得分70.

3673466	张晨阳	张晨阳	出行计划	05-13 16:07	429B	CPP11	运行超时	70	运行超时	3.781MB

C++

```
#include<bits/stdc++.h>
# define N 100005
using namespace std;
int main()
{
        int n, m, k, q;
        cin >> n >> m >> k;
        int t, c, op[2 * N];
        for (int i = 0; i < n; i++) {
                cin >> t >> c;
                op[2 * i] = t - k;
                op[2 * i + 1] = t - k - c + 1;
        }
        for (int i = 0; i < m; i++) {
                cin >> q;
                int count = 0;
                for (int j = 0; j < 2 * n; j += 2)
                        if (q <= op[j] && q >= op[j + 1])
                                count++;
                cout << count << endl;
        }
        return 0;
}
```

解法二：

用一个 $vector$ 记录每个计划可以进入的时刻，对于每个计划：数组$[t-c+1, t]$的位置$+1$。
直接查询数组$[q+k]$位置的值即为所求。
注意：$vector$ 需要开到$3N$ 大小，$q+k$可到3×10^5

时间复杂度$O((n+m)logn)$，空间复杂度$O(n)$，期望得分100

3674006	张晨阳	张晨阳	出行计划	05-13 19:20	538B	CPP14	正确	100	937ms	4.113MB

C++

```
#include<iostream>
#include<vector>
# define N 300005
using namespace std;

void timesum(vector<int>& v,int a,int b)
{
        for (vector<int>::iterator it = v.begin() + a; it != v.begin() + b +
1; it++)
                (*it)++;
}

int main()
{
        int n, m, k, q;
        cin >> n >> m >> k;
        int t, c;
        vector<int> time;
        time.assign(N, 0);
        for (int i = 0; i < n; i++) {
                cin >> t >> c;
                if (t - c + 1 >= 0)
                        timesum(time, t - c + 1, t);
                else
                        timesum(time, 0, t);
        }
        for (int i = 0; i < m; i++) {
                cin >> q;
                cout << time[q + k] << endl;
        }

        return 0;
}
```

题目 3：计算资源调度器

题目描述：

三种标准需求，每次三选若干。

- 计算节点亲和性
  计算任务必须在**指定可用区**上运行。
- 计算任务亲和性
  计算任务必须和**指定应用**的计算任务在同一可用区上运行。
- 计算任务反亲和性（必须满足/尽量满足）
  计算任务不能和**指定应用**的计算任务在同一个计算节点上运行。

一旦选择了一个计算节点，就固定下来不再变动， 并且在此后的选择中，不再考虑这个计算任务的要求。
对每个计算任务，选择计算节点的方法是：1. 过滤阶段 2. 排序阶段

- f_i : 表示要接连启动 f_i 个所属应用和要求相同的计算任务，其中 f_i>0；
- a_i ： 表示这 a_i 个计算任务所属应用的编号；
- na_i ： 表示计算节点亲和性要求。当 $na_i = 0$ 时，表示没有计算节点亲和性要求；否则表示要运行在编号为 na_i 的可用区内的计算节点上；
- pa_i ： 表示计算任务亲和性要求。当 $pa_i = 0$ 时，表示没有计算任务亲和性要求；否则表示必须和编号为 pa_i 的应用的计算任务在同一个可用区运行；
- paa_i: 表示计算任务反亲和性要求。当 paa_i=0 时，表示没有计算任务反亲和性要求；否则表示不能和编号为 paa_i 的应用的计算任务在同一个计算节点上运行；
- $paar_i$ ： 表示计算任务亲和性要求是必须满足还是尽量满足，当 $paa_i = 0$ 时，$paar_i$ 也一定为 0；否则 $paar_i$=1 表示“必须满足”，$paar_i$=0 表示“尽量满足”。
$0 < m \leq n \leq 1000, 0 < g \leq \sum_{i=1}^{g} f_i \leq 2000, 0 < A_{max} \leq 10^9$

题解：

解法一：

构造结构$node$:

- id表示节点编号，$zone$表示节点所在可用区，num表示节点运行任务数量。
- 用$bool$数组app表示每个节点运行的应用，$app[i] = 1$表示应用 i 运行。
用二维$bool$数组 $apply[a][b]$ 表示可用区a运行应用b。
数组$C1[i]$ 记录是否满足节点亲和性和任务亲和性，数组$C2[i]$ 记录是否满足反亲和性。
每个$func$ 函数调用前，将数组$C1$ 和$C2$ 初始化为0。

时间复杂度$O(n^2)$, 空间复杂度$O(A^2)$, 期望得分50。
（对于一半的数据，A_{max}可达10^9, 数组下标越界，导致运行错误）

3686416	张晨阳	张晨阳	计算资源调度器	05-18 11:22	1.627KB	CPP11	运行错误	50	15ms	4.074MB

C++

```cpp
#include<bits/stdc++.h>
#define N 1005
using namespace std;

int n, m, g;
int C1[N], C2[N];
struct node {
	int id;				//节点编号
	int zone;			//节点所在区
	int num;			//节点运行任务数量
	bool app[N] = { 0 };//节点运行应用
}node[N];
bool apply[N][N];		//可用区运行应用

void jie(int na)
{
	if (!na)
		for (int i = 1; i <= n; i++)
			C1[i]++;
	else
		for (int i = 1; i <= n; i++)
			if (na == node[i].zone)
				C1[i]++;
}

void ren(int pa)
{
	if (!pa)
		for (int i = 1; i <= n; i++)
			C1[i]++;
	else
		for (int i = 1; i <= n; i++)
			if (apply[node[i].zone][pa])
				C1[i]++;
}

void fan(int paa)
{
	if (!paa)
		for (int i = 1; i <= n; i++)
			C2[i]++;
	else
		for (int i = 1; i <= n; i++)
			if (!node[i].app[paa])
				C2[i]++;
}

void func(int a,int na,int pa,int paa,int paar)
```

```
{
        jie(na);
        ren(pa);
        fan(paa);
        int mmin = N;
        int id = 0;
        for (int i = 1; i <= n; i++)
                if (C1[i] == 2 && C2[i] == 1)
                        if (node[i].num < mmin) {
                                mmin = node[i].num;
                                id = i;
                        }
        if(id==0 && paar==0)
                for(int i = 1; i <= n; i++)
                        if (C1[i] == 2)
                                if (node[i].num < mmin) {
                                        mmin = node[i].num;
                                        id = i;
                                }
        cout << id << " ";
        if (id) {
                node[id].num++;
                node[id].app[a] = 1;
                apply[node[id].zone][a] = 1;
        }
}

int main()
{
        cin >> n >> m;
        int L;
        for (int i = 1; i <= n; i++) {
                cin >> L;
                node[i].id = i;
                node[i].zone = L;
                node[i].num = 0;
        }
        cin >> g;
        for (int i = 0; i < g; i++) {
                int f, a, na, pa, paa, paar;
                cin >> f >> a >> na >> pa >> paa >> paar;
                while (f--) {
                        fill(C1, C1 + N, 0);
                        fill(C2, C2 + N, 0);
                        func(a, na, pa, paa, paar);
                }
                cout << endl;
        }
```

```
        return 0;
}
```

解法二:

对解法一进行空间优化:
使用set 容器替换一维$bool$ ， 若运行应用a ， 则$set.count(a) > 0$
使用map 容器替换二维数组，则无需开辟$A * A$的数组; $map[a][b]$ 表示可用区a 运行应用b, 运行则值为1

时间复杂度$O(n^2 log\ n)$, 空间复杂度$O(n)$, 期望得分100.

3686695	张晨阳	张晨阳	**计算资源调度器**	05-18 12:53	1.590KB	CPP14	正确	100	46ms	3.226MB

C++

```cpp
#include<bits/stdc++.h>
#define N 1005
using namespace std;

int n, m, g;
int C1[N], C2[N];
struct node {
        int id;                         //节点编号
        int zone;                       //节点所在区
        int num;                        //节点运行任务数量
        set<int> app;           //节点运行应用
}node[N];
map<int,int> apply[N];

void jie(int na)
{
        if (!na)
                for (int i = 1; i <= n; i++)
                        C1[i]++;
        else
                for (int i = 1; i <= n; i++)
                        if (na == node[i].zone)
                                C1[i]++;
}

void ren(int pa)
{
        if (!pa)
                for (int i = 1; i <= n; i++)
                        C1[i]++;
        else
                for (int i = 1; i <= n; i++)
                        if (apply[node[i].zone][pa])
                                C1[i]++;
}

void fan(int paa)
{
        if (!paa)
                for (int i = 1; i <= n; i++)
                        C2[i]++;
        else
                for (int i = 1; i <= n; i++)
                        if (!node[i].app.count(paa))
                                C2[i]++;
}

void func(int a,int na,int pa,int paa,int paar)
```

```
{
        jie(na);
        ren(pa);
        fan(paa);
        int mmin = N;
        int id = 0;
        for (int i = 1; i <= n; i++)
                if (C1[i] == 2 && C2[i] == 1)
                        if (node[i].num < mmin) {
                                mmin = node[i].num;
                                id = i;
                        }
        if(id==0 && paar==0)
                for(int i = 1; i <= n; i++)
                        if (C1[i] == 2)
                                if (node[i].num < mmin) {
                                        mmin = node[i].num;
                                        id = i;
                                }
        cout << id << " ";
        if (id) {
                node[id].num++;
                node[id].app.insert(a);
                apply[node[id].zone][a] = 1;
        }
}

int main()
{
        cin >> n >> m;
        int L;
        for (int i = 1; i <= n; i++) {
                cin >> L;
                node[i].id = i;
                node[i].zone = L;
                node[i].num = 0;
        }
        cin >> g;
        for (int i = 0; i < g; i++) {
                int f, a, na, pa, paa, paar;
                cin >> f >> a >> na >> pa >> paa >> paar;
                while (f--) {
                        fill(C1, C1 + N, 0);
                        fill(C2, C2 + N, 0);
                        func(a, na, pa, paa, paar);
                }
                cout << endl;
        }
```

```
        return 0;
}
```

题目 4：通信系统管理

题目描述：

n 台计算机，u, v, x, y 表示机器u 和v 的每日可用额度增大x MB/day，持续y 天.
定义每台机器的“通信主要对象”：
当前时刻与该机器的每日可用额度最大的机器（如果有并列，则取其中编号最小的机器）；
如果一台机器与任何机器的每日可用额度均为 0 ，则称其为“通信孤岛”，并认为其没有“通信主要对象”；如果两台机器 x 和 y 互为“通信主要对象”，则称它们是一个“通信对”。

- 每天开始时，先接受若干个额度申请，你需要依次处理这些申请；
- 接收若干个查询某台机器的“通信主要对象”的请求；求出此时的“通信孤岛”和“通信对”各有多少。
  $1 \leq n, m \leq 10^5, 1 \leq A, B \leq 2 \times 10^5, 1 \leq u, v \leq n, 1 \leq x \leq 10^9, 1 \leq y \leq m$

题解:

解法一:

每日额度有效期可以看成一个激活与反激活的过程，反激活即在该天将额度减去。
维护额度最大值，用set 以额度为关键字作降序排列，把x 和v 变成一个$pair$ 压入到set 里，再开一个数据结构记录u，v 时x 的值；保证可以通过v 访问，也能通过v 求最大值。
注：无论给谁加额度，都是给u，v 两个同时加。
在每次新增每日额度时，维护通信孤岛和通信对的数量。

时间复杂度$O(nlog\,n)$, 期望得分100

3694512	张晨阳	张晨阳	通信系统管理	05-20 20:11	1.916KB	CPP14	正确	100	1.718s	49.79MB

C++

```cpp
#include<bits/stdc++.h>
#define maxn 100010
using namespace std;

struct node {
        long long value;
        int to;
        node(){}
        node(long long value,int to) : value(value), to(to){}

        bool operator<(const node& d) const {
                return value == d.value ? to<d.to : value>d.value;
        }
};
struct info {
        int u, v, x;
        info(int u,int v, int x) : u(u), v(v), x(x){}
};
set<node> d[maxn];
map<pair<int, int>, long long> save;
vector<info> deactive[maxn];   //反激活
int p_value, q_value;

//检查一个点是否为孤岛
int check_p(int id) {
        return d[id].begin() == d[id].end() || d[id].begin()->value == 0;
}

//检查一个点是否包含一个通讯对
int check_q(int x) {
        if (check_p(x))
                return 0;
        int y = d[x].begin()->to;
        return (!check_p(y)) && d[y].begin()->to == x;
}

void work(int u, int v, int x) {
        long long orgvalue = save[{u, v}];
        save[{u, v}] += x;
        //处理孤岛数量
        p_value -= check_p(u);
        //处理通讯对数量
        q_value -= check_q(u);

        //删除旧的
        node org(orgvalue, v);
        d[u].erase(org);
```

```
        //插入新的
        d[u].emplace(save[{u, v}], v);

        //处理孤岛数量
        p_value += check_p(u);
        //处理通讯对数量
        q_value += check_q(u);
}

int main()
{
        ios::sync_with_stdio(false);
        int n, m;
        cin >> n >> m;
        p_value = n;
        q_value = 0;
        for (int i = 0; i < m; i++) {
                //处理过期额度
                for (const auto& x : deactive[i]) {
                        work(x.u, x.v, -x.x);
                        work(x.v, x.u, -x.x);
                }
                int k;
                cin >> k;
                //输入额度
                while (k--) {
                        int u, v, x, y;
                        cin >> u >> v >> x >> y;
                        if (i + y <= m)
                                deactive[i + y].emplace_back(u, v, x);
                        work(u, v, x);
                        work(v, u, x);
                }
                //输入通信主要对象
                int l;
                cin >> l;
                while (l--) {
                        int id;
                        cin >> id;
                        if (check_p(id))
                                cout << 0 << "\n";
                        else
                                cout << d[id].begin()->to << "\n";
                }
                int p, q;
                cin >> p >> q;
                //查询孤岛数量
                if (p)
                        cout << p_value << "\n";
```

```
                //查询通讯对数量
                if (q)
                        cout << q_value << "\n";
        }
        return 0;
}
```

题目 5：博弈论与石子合并

题目描述：

n 堆石子，第i 堆有a_i 个石子
三种操作：

1. 合并两堆相邻的石子
2. 扔掉目前最靠左的一堆石子
3. 扔掉目前最靠右的一堆石子
   小 c 希望这堆石子尽量少，小 z 则希望这堆石子尽量多。
   $k = 0$ 表示小 c 先操作，$k = 1$ 表示小 z 先操作。
   问最后这堆石子的大小是多少。
   $1 \le n \le 10^5, 0 \le k \le 1, a_i > 0, \sum_{i=1}^{n} a_n \le 10^9$

题解：

解法一：

- 小 c 先手+奇数堆和小 z 先手+偶数堆等价，即最后一次都是合并剩余两堆；
  $n = 2k + 1, k = 1, 2...$，小 c 移去左右最大的一堆，小 z 合并a_k, a_{k+1} 两堆。则最终都是$k + 1$ 堆的和。
- 小 c 先手+偶数堆和小 z 先手+奇数堆等价，即最后一次都是移去最大堆；
  小 c 操作$\frac{n}{2}$ 次，若找到一个石子数x，存在大于$\frac{n}{2}$ 堆的x，则最后总能留下x。x 即为所求。
  用二分法求x。

时间复杂度$O(nlog\,n)$，期望得分100

3695367	张晨阳	张晨阳	博弈论与石子合并	05-21 00:44	914B	CPP14	正确	100	31ms	3.191MB

C++

```cpp
#include<bits/stdc++.h>
#define N 100005
using namespace std;
int n, k;
int a[N];
int work1() {
	int mid = n / 2;
	int ans = 0;
	for (int i = 0; i <= mid; i++)
		ans += a[i];
	int temp = ans;
	for (int i = mid + 1; i < n; i++) {
		temp = temp + a[i] - a[i - mid - 1];
		ans = min(ans, temp);
	}
	return ans;
}

bool check(int x) {
	int sum = 0, count = 0;
	for (int i = 0; i < n; i++) {
		sum += a[i];
		if (sum >= x) {
			sum = 0;
			count++;
		}
	}
	return count > n / 2;
}

int work2(int l, int r) {
	int ans = 0;
	while (l <= r) {
		int mid = (l + r) / 2;
		if (check(mid)) {
			ans = mid;
			l = mid + 1;
		}
		else
			r = mid - 1;
	}
	return ans;
}

int main()
{
	ios::sync_with_stdio(false);
	cin >> n >> k;
```

```
        int minx = 1e9, sum = 0;
        for (int i = 0; i < n; i++) {
                cin >> a[i];
                sum += a[i];
                minx = min(minx, a[i]);
        }
        if ((n + k) % 2)
                cout << work1();
        else
                cout << work2(minx, sum);
        return 0;
}
```