

5.29 测验参考答案

课后自己洛谷自测

1、P1802

01 背包题，但是注意精力的取值要包含 0（即 j 的取值要包含 0），因为就算精力为 0，采不了这种药，也有价值。此外要注意输出是最高价值*5。

```
#include<iostream>
using namespace std;
int n,x;
long long lose[1010],win[1010],use[1010];
long long dp[1010];

int main(){
    cin >> n >> x;
    for(int i = 1; i <= n; i++){
        cin >> lose[i] >> win[i] >> use[i];
    }
    for(int i=1; i<=n; i++){//遍历草药
        for(int j = x; j >= 0; j--){ // j必须大于等于0，因为就算采不了药也有价值
            if(j >= use[i]){
                dp[j] = max(dp[j] + lose[i], dp[j-use[i]] + win[i]);
            }
            else{
                dp[j] = dp[j] + lose[i];//不采药也有价值
            }
        }
    }
    cout << dp[x]*5 << endl; //最后结果要乘以5
    return 0;
}
```

2、P3395

广度优先搜索，定义一个 mapp 数组用来标记路障，再定义一个 vis 数组用来标记走过的路，下一个点可以走的条件是之前没有走过并且当前没有路障。在结构体里再定义一个变量 t 用来记录时间，注意是当前秒结束之后，某个点才会被摆上路障，所以更新 mapp 数组的时候 t 要减 1。

```

#include<bits/stdc++.h>
using namespace std;
#define N 2020
int n,k,mapp[N][N],a[N],b[N]; //map数组是地图, a和b数组保存障碍坐标
bool vis[N][N],flag; //vis是标记数组, 用来标记走过的坐标
int dx[4]={0,0,1,-1};
int dy[4]={1,-1,0,0};
struct node {
    int x,y,t; //t用来记录时间
};
queue<node> q;

void bfs(int x,int y,int t) {
    node head,tail;
    head.x=x;
    head.y=y;
    head.t=t;
    q.push(head);
    while(!q.empty()) {
        head = q.front();
        q.pop();
        if(head.x==n && head.y==n) { //走到(n,n)
            flag=true;
            return ;
        }
        mapp[a[head.t-1]][b[head.t-1]]=1; //在那一秒结束后, (x,y)才被摆上路障, 所以这里是相应的时间-1

        for(int i=0;i<4;i++) { //四个方向搜索
            int tx=head.x+dx[i];
            int ty=head.y+dy[i];
            if(tx>=1 && tx<=n && ty>=1 && ty<=n && mapp[tx][ty]==0 && vis[tx][ty]==false) {
                vis[tx][ty]=true; //标记该点
                tail.x=tx;
                tail.y=ty;
                tail.t=head.t+1; //时间+1
                q.push(tail);
            }
        }
    }
}

int main() {
    cin >> k;
    while(k--) {
        cin >> n;
        for(int i=1;i<=2*n-2;i++) {
            cin >> a[i] >> b[i];
        }
        flag=false; //每次循环, flag初始化为false
        memset(vis,false,sizeof(vis)); //每次循环, 标记数组初始化为false
        memset(mapp,0,sizeof(mapp)); //每次循环, 地图全部初始化为0
        vis[1][1]=true; //起点初始化

        bfs(1,1,1);
        if(flag) {
            cout << "Yes" << endl;
        }
        else {
            cout << "No" << endl;
        }
    }
    return 0;
}

```

3、P1843

二分答案。用 mid 来表示除掉冰柱所用的时间, 如何判断这个时间不可行: 依次比较某根冰柱自然融化所需的时间和 mid, 如果 mid 大于等于某根冰柱自然融化所需的时间, 则不需要用到加热器, 如果 mid 小于某根冰柱自然融化所需的时间, 则需要用到加热器, 计算每根冰柱需要用到加热器的时间 (注意**向上取整**的情况), 求和, 再跟 mid 进行比较, 如果用到加热器的总时间不超过 mid, 则此 mid 可行。然后用二分法去更新 mid 逼近最小值即可。

```

#include<iostream>
using namespace std;
int n, a, b;
int w[500010];

bool check(int num){
    int cnt = 0;
    for(int i=1; i<=n; i++){
        if(w[i] > num*a){
            int x = w[i] - num*a; //x表示需要加热器烘干的体积部分
            if(x%b == 0){
                cnt += x/b; //统计每根冰柱需要用到加热器的时间
            }
            else{
                cnt += x/b + 1; //向上取整
            }
        }
    }
    if(cnt > num){
        return false;
    }
    else{
        return true;
    }
}

```

```

int ErFen(){
    int mid, ans, l, r;
    l = 0;
    r = 5e5;
    while(l <= r){
        mid = (l+r)/2;
        if(check(mid)){
            r = mid - 1;
            ans = mid;
        }
        else{
            l = mid + 1;
        }
    }
    return ans;
}

```

```

int main(){
    cin >> n >> a >> b;
    for(int i=1; i<=n; i++){
        cin >> w[i];
    }
    cout << ErFen() << endl;
    return 0;
}

```

4、P3205

区间 DP。将大区间的初始队形方案数转化为小区间的方案数，而当区间从小到大扩展的时候（即插进来一个人的时候），可能从左插入也可能从右插入，所以要分两种情况讨论。定义 $f[i][j]$ 表示可以组成理想队形 $[i, j]$ 区间的初始队形数，分两种情况讨论，再定义： $f[i][j][0]$ 表示排在第 i 位的人从左边插入的方案数， $f[i][j][1]$ 表示排在第 j 位的人从右边插入的方案数。

(1) 假如当前的人是插入到第 i 位，是从左边插入的，那说明他比在他前面一次插入的人矮；而在他前面一次插入的人，要么插到了最左边（第 $i+1$ 位），要么插到了最右边（第 j 位）。

（也就是说 $f[i][j][0]$ 一定继承的是 $f[i+1][j][0]$ 或者 $f[i+1][j][1]$ ）

(2) 假如当前的人是插入到第 j 位，是从右边插入的，那么说明他比在他前面一次插入的人高；而在他前面一次插入的人，要么插到最左边（第 i 位），要么插到最右边（第 $j-1$ 位）。

（也就是说 $f[i][j][1]$ 一定继承的是 $f[i][j-1][0]$ 或者 $f[i][j-1][1]$ ）

注意初始化一个人排成一行只有一种方案，从左插入或者从右插入，只能选其一。

```
using namespace std;
const int maxn = 1e3+5;
const int mod = 19650827;
int n;
int h[maxn];
int f[maxn][maxn][2];
int main(){
    cin >> n;
    for(int i=1; i<=n; i++){
        cin >> h[i];
        f[i][i][1] = 1; // 写 f[i][i][0]=1 也行，但只能写一个为1
    }
    for(int len=2; len<=n; len++){
        for(int i=1, j=i+len-1; j<=n; i++, j++){
            if(h[i] < h[i+1]) f[i][j][0] += f[i+1][j][0];
            if(h[i] < h[j]) f[i][j][0] += f[i+1][j][1];

            if(h[j] > h[i]) f[i][j][1] += f[i][j-1][0];
            if(h[j] > h[j-1]) f[i][j][1] += f[i][j-1][1];

            f[i][j][0] %= mod;
            f[i][j][1] %= mod;
        }
    }
    cout << (f[1][n][0] + f[1][n][1]) % mod << endl;
    return 0;
}
```

附加题：P4290

区间 DP。定义 $dp[i][j][k]$ 表示区间 $[i, j]$ 可以由 k 转化而来。而在对输入预处理的时候，定义 $f[i][j][k]$ 表示 i 可以由 j, k 代替。在合并区间的时候，除了要枚举区间长度、起点、断点、终点外，还要枚举左子区间合并后转化成的字母、右子区间合并后转化成的字母、总区间合并后转化成的字母。同时还要加上可以合并的条件：左子区间可合并转化、右子区间可合并转化、左/右子区间的合并字母和总区间的合并字母之间的可替代性（即 $f[i][j][k]$ 数组）。为了方便表示，可以将字母转化为数字表示。

```
#include<bits/stdc++.h>
using namespace std;
int dp[205][205][5], f[205][205][5], num[5], n=0;
char s[205], c[2];

int add(char x) //为了方便统计，字母转化为数字表示
{
    if(x=='W') return 1;
    if(x=='I') return 2;
    if(x=='N') return 3;
    if(x=='G') return 4;
}

int main()
{
    for(int i=1; i<=4; i++) {
        cin >> num[i];
    }
    for(int i=1; i<=4; i++) {
        for(int j=1; j<=num[i]; j++)
        {
            cin >> c[0] >> c[1];
            f[i][add(c[0])][add(c[1])]=1; //将字母之间的可替代性表示出来
        }
    }
    cin >> s+1;

    for(n=1; s[n]; n++); //计算字符数组的长度，最后要减1
    n--;

    for(int i=1; i<=n; i++){
        dp[i][i][add(s[i])]=1; //初始化：自己可以转化为自己
    }

    for(int len=1; len<=n; len++) //枚举区间长度
        for(int l=1; l+len-1<=n; l++) //枚举起点
        {
            int r=l+len-1;
            for(int k=1; k<=4; k++) //枚举断点
                for(int i=1; i<=4; i++) //枚举最后合并成的字母
                    for(int j=1; j<=4; j++) //枚举左子区间合并成的字母
                        for(int g=1; g<=4; g++) //枚举右子区间合并成的字母
                            if(f[i][j][g]&&dp[l][k][j]&&dp[k+1][r][g]) //可以合并的条件
                                dp[l][r][i]=1;
        }

    bool flag=0;
    if(dp[1][n][1]) {flag=1; cout << "W";}
    if(dp[1][n][2]) {flag=1; cout << "I";}
    if(dp[1][n][3]) {flag=1; cout << "N";}
    if(dp[1][n][4]) {flag=1; cout << "G";}
    if(!flag) cout << "The name is wrong!";
    return 0;
}
```