# 算法第四轮考核

19计科4邓启裕

杭电ID：GDQYIF

洛谷ID：dddif

VJ ID：QYIF

CF ID：QYIF

题数：35+4+5+7(算法学习+CF三场比赛加补题【有些题暂时还没弄懂】)

刷题有些是同一题但用了不同的方法

# 1、基础算法：

## 1）搜索：DFS&BFS

上轮考核我曾经刷过搜索的提单，但是dfs和bfs都是背模板来写的，而且有时，要么找不全边界条件，要么就是写错条件，筛选方式不对。这次考核重新学习了一下这个模块。

### a）DFS

**典型例题：P1706 全排列问题**

#include<iostream>

#include<iomanip>

using namespace std;

const int N = 20;

int n;

int ans[N];

int used[N];

void dfs(int x) {

if (x > n) {

for (int j = 1; j <= n; j++)

cout << setw(5) << ans[j];

cout << endl;

return;

}

for (int i = 1; i <= n; i++)

if (!used[i]) {

ans[x] = i;

used[i] = 1;

dfs(x + 1);

used[i] = 0;

}

}

int main()

{

cin >> n;

dfs(1);

}

思路：全排列算是dfs最经典的题之一了。dfs用的是栈结构，可以用递归来写。而递归基本思路就是设置一个递归结束的条件（在本题中就是找齐数字），而递归部分基本思路就是搜索当前行，符合条件就继续往下搜索，并记录这个数已使用。搜网以后回溯，把这个数重新标记为未使用，继续往下找。

### b)BFS

**经典例题：P1443 马的遍历**

#include<iostream>

#include<algorithm>

#include<queue>

#include<cstring>

#include<iomanip>

using namespace std;

const int N = 401;

int op[8][2] = { {2,1},{1,2} ,{-1,2}, {-2,1} ,{-1,-2} ,{-2,-1} ,{1,-2} ,{2,-1} };

int a[N][N], m, n, sx, sy;

bool used[N][N];

struct node { int x, y; };

bool inMap(int x, int y)

{

return x >= 1 && x <= n && y >= 1 && y <= m;

}

void bfs(int x, int y)

{

used[x][y] = 1;

queue<node> q;

a[x][y] = 0;

q.push({ x, y });

while (!q.empty())

{

node cur = q.front();

for (int i = 0; i < 8; i++)

{

int curX = cur.x + op[i][0];

int curY = cur.y + op[i][1];

if (inMap(curX, curY) && !used[curX][curY])

{

used[curX][curY] = 1;

a[curX][curY] = a[cur.x][cur.y] + 1;

q.push({ curX,curY });

}

}

q.pop();

}

}

int main()

{

cin >> n >> m >> sx >> sy;

memset(a, -1, sizeof(a));

bfs(sx, sy);

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= m; j++)

cout << setiosflags(ios::left) << setw(5) << a[i][j];

cout << endl;

}

return 0;

}

思路：BFS的思路与DFS不一样，BFS用到的数据结构是队列，基本实现思路是先创建一个队列，每次取出队列中的第一个元素，对这个元素进行操作，或者检测这个元素是否符合条件，然后把该元素的紧邻压入队列中。直到队列里没有元素为止。而这道题中，首先创建队列，然后对马的走位的几个位置进行检测，如果不超出地图就记录为已检测，并把该点的临点（即马的其他没走过的走位）压入队列，以此类推直到队列为空。

## 强化练习

#### 1） 棋盘问题

代码：

#include<iostream>

#include<string.h>

using namespace std;

char map[10][10];

int n, k, ans = 0, used[10];

void dfs(int level, int chosen) {

if (chosen == k) { ans++; return; }

if (level == n)return;

for (int lie = 0; lie < n; lie++) {

if (map[level][lie] == '#' && !used[lie]) {

used[lie] = 1;

dfs(level + 1, chosen + 1);

used[lie] = 0;

}

}

dfs(level + 1, chosen);

}

int main() {

while (cin >> n >> k && (n != -1 && k != -1)) {

memset(used, 0, sizeof(used));

ans = 0;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

cin >> map[i][j];

dfs(0, 0);

cout << ans << endl;

}

}

思路：这道题有点坑，坑就在于不是所有可选点都选得上，而且选点时候还可能跳着选。“dfs(level + 1, chosen)“这一段代码我是半天没想出来，一开始我还用for循环调用dfs后来错了，才意识到循环调用dfs是连续的而选点是可以不连续。所以在每次dfs选点时，还要考虑跳过该点的情况。总体思路就是dfs找满足点，找到后再分成宣和不选两种情况，然后就ok了。（上来就是一个下马威，还简单搜索，我太菜了555）

#### 2）P1451 求细胞数量

#include<iostream>

#include<algorithm>

#include<queue>

#include<cstring>

#include<iomanip>

using namespace std;

const int N = 105;

int op[4][2] = { {1,0}, {-1,0} ,{0,1} ,{0,-1} };

int m, n, ans;

char map[N][N];

bool used[N][N];

struct node { int x, y; };

bool isNum(int x, int y)

{

return map[x][y] >= '1' && map[x][y] <= '9';

}

void bfs(int x, int y)

{

used[x][y] = 1;

queue<node> q;

q.push({ x, y });

while (!q.empty())

{

node cur = q.front();

for (int i = 0; i < 4; i++)

{

int curX = cur.x + op[i][0];

int curY = cur.y + op[i][1];

if (isNum(curX, curY) && !used[curX][curY])

{

used[curX][curY] = 1;

q.push({ curX,curY });

}

}

q.pop();

}

ans++;

}

int main()

{

cin >> n >> m;

for (int i = 1; i <= n; i++)

for (int j = 1; j <= m; j++)

cin >> map[i][j];

for (int i = 1; i <= n; i++)

for (int j = 1; j <= m; j++)

if (!used[i][j] && isNum(i, j))

bfs(i, j);

cout << ans;

return 0;

}

思路：这道题我是用bfs做的，创建队列然后判断是否是1到9是就标记为已搜索，然后继续搜，搜到队列为空为止。搜完这一块后ans++。后面也的没搜过的也是这样子操作。知道全都搜了一遍为止。

#### 3) HDU – 1241 Oil Deposits

#include <iostream>

using namespace std;

const int N = 101;

int op[8][2] = { {1,0}, {-1,0}, {0,1} , {0,-1} , {1,-1} , {1,1} , {-1,-1} , {-1,1} };

char map[N][N];

int m, n;

bool over(int x, int y)

{

return (x < 0) || (y < 0) || (x > n - 1) || (y > m - 1);

}

void dfs(int a, int b)

{

if (map[a][b] != '@' || over(a, b))return;

map[a][b] = '\*';

for (int i = 0; i < 8; i++)

dfs(a + op[i][0], b + op[i][1]);

return;

}

int main()

{

int ans;

while (cin >> n >> m)

{

if (m == 0 && n == 0)break;

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++)

cin >> map[i][j];

ans = 0;

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++)

if (map[i][j] == '@')

{

dfs(i, j);

ans++;

}

cout << ans << endl;

}

return 0;

}

思路：这道题和上一道有点像，这道题我用dfs做的。思路差不多目标也就是搜一个点然后往八个方向搜直到搜不到为止（每个点搜完都要标为已搜索），执行一次搜索后ans++，直到所有点都搜完为止。

### **专题总结：**

这个专题主要是学了搜索，搜索主要有DFS和BFS两种算法，据说（我也不知道是不是）数据比较大时最好用BFS。我个人感觉，通常要求所有解，或者要把所有历遍一遍的可以用DFS，而如果要找最优解，最短路（没权那种）就最后用BFS。

# 2.数论

## 1）快速幂

#include<iostream>

#include<algorithm>

using namespace std;

int main() {

long long ans = 1, b, p, k;

cin >> b >> p >> k;

cout << b << "^" << p << " mod " << k << "=";

while (p > 0) {

if (p % 2 == 1)

ans = ans \* b % k;

b = b \* b % k;

p = p / 2;

}

cout << ans % k;

return 0;

}

思路：就是把幂不断的二分，每次把b自乘以此并执行求余数的操作，而多出来的部分直接放到答案ans中去，最后p为1时，ans就与b相乘，最后再把答案求一次余即可。

## 2）矩阵快速幂

#include<iostream>

#include<algorithm>

#include<cstring>

#define ll long long

using namespace std;

ll tmp[1][1];

ll ans[1][1];

void multi(ll a[1][1], ll b[1][1], int n, int t)

{

memset(tmp, 0, sizeof tmp);

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

for (int k = 0; k < n; k++)

tmp[i][j] = ((a[i][k] \* b[k][j]) % t + tmp[i][j]) % t;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

a[i][j] = tmp[i][j] % t;

}

void Pow(ll a[1][1], int n, int k, int p)

{

memset(ans, 0, sizeof ans);

for (int i = 0; i < n; i++) ans[i][i] = 1;

while (p > 0)

{

if (p % 2 == 1)

multi(ans, a, n, k);

multi(a, a, n, k);

p /= 2;

}

}

int main() {

ll a[1][1], p, k;

cin >> a[0][0] >> p >> k;

cout << a[0][0] << "^" << p << " mod " << k << "=";

Pow(a, 1, k, p);

cout << ans[0][0] % k << endl;

return 0;

}

思路：其实和快速幂差不多，只是变成了矩阵了而已，同样的模板题，这次用矩阵做了一下，首先把这个数b看成一个矩阵（一个1\*1的矩阵），然后先写一个矩阵相乘的函数，再写一个快速快速幂的函数（这里我不得不吐槽一下这个求余，太烦了，求余计算要加在好几个地方才能确保计算出来的答案是正确的）。快速幂函数的写法与上面大致相同，这里就不重复了。

## 3）辗转相除法：

### a)欧几里得：

UVA11417 GCD

#include<iostream>

using namespace std;

int gcd(int a, int b) {

if (b == 0)return a;

return gcd(b, a % b);

}

int main() {

int n;

while (cin >> n)

{

if (!n)break;

int ans = 0;

for (int i = 1; i < n; i++)

for (int j = i + 1; j <= n; j++)

ans += gcd(i, j);

cout << ans << endl;

}

return 0;

}

思路：辗转相除法就是两数大除以小求余不断重复直到余数为，则此时的除数即为gcd。

递归写gcd真的好简洁。

### b)拓展欧几里得：

P5656 【模板】二元一次不定方程(exgcd)

#include<iostream>

#include<stdio.h>

using namespace std;

#define ll long long

ll x, y;

ll gcd(ll a, ll b)

{

return b == 0 ? a : gcd(b, a % b);

}

void exgcd(ll a, ll b, ll& x, ll& y)

{

if (b == 0)

{

x = 1;

y = 0;

return;

}

ll temp;

exgcd(b, a % b, x, y);

temp = x;

x = y;

y = temp - (a / b) \* y;

return;

}

int main()

{

cin.tie(0);

int n;

cin >> n;

for (int i = 0; i < n; i++)

{

ll a, b, c, xMax, xMin, yMax, yMin, num = 0, x = 0, y = 0;

cin >> a >> b >> c;

ll g = gcd(a, b);

if (c % g != 0)

{

printf("-1\n");

}

else

{

a /= g;

b /= g;

c /= g;

exgcd(a, b, x, y);

x \*= c;

y \*= c;

if (x > 0 && x % b != 0)

xMin = x % b;

else

xMin = x % b + b;

yMax = (c - xMin \* a) / b;

if (y > 0 && y % a != 0)

yMin = y % a;

else

yMin = y % a + a;

xMax = (c - yMin \* b) / a;

if (xMax > 0)

num = (xMax - xMin) / b + 1;

if (num == 0)

printf("%lld %lld\n", xMin, yMin);

else

printf("%lld %lld %lld %lld %lld\n", num, xMin, yMin, xMax, yMax);

}

}

return 0;

}

思路：这道题真的写了好久呀！！！先是学习了一下什么是拓展欧几里得算法（找不到视频，看了几个博客才懂）然后去了解了一下怎么用exgcd解二元一次不定方程，然后分类讨论弄了半天出不来一个正确结果，最后还要解决加速输入输出的问题。最后看了下题解才弄得出来。说说总体思路。首先是写出gcd和exgcd的代码，gcd上面写过了，exgcd与gcd的思路差不多主要是解决ax+by=gcd(a,b)这种方程的一个特解。而二元不定方程ax+by=c的解决思路主要是先把这个方程转换称a2x+b2y=gcd(a,b)的形式，即两边同乘gcd(a,b)/c得到a2x+b2y=gcd(a,b)的方程后再利用exgcd求出其中的一个特解。再把方程转换回原来的形式，即把exgcd求出来的解分别除回gcd(a,b)/c就可以得到ax+by=c的一个特解。求得特解以后就可以求通解了，xt​=x0​+b2t，yt=y0​−a2t​。接下来就是最伤脑经的时候了--分类讨论.

首先是无解的情况，当c不是gcd(a,b)的倍数时，就无整数解了直接输出-1，而讨论其他情况时，值得注意的是，x的取值与y的取值是成反比的，所以x取得越小y取得越大，即我们可以通过x的最大值得到y的最小值，反之也是。接下来进行分类讨论（这里看了题解才写得出来）若x>0且x%b!=0，x的最小正整数解就等于x%b，否则则需要在x%b的基础上加b。反之也是。如果无整数解不必计算num（整数解的个数）num=0此时执行第一个输出。否则有多个解，执行第二种输出。最后处理一个问题（输入输出的时间问题），一开始直接cin，cout结果超时了，上网找了找，找到了cin.tie(0);这个东西可以提速于是用了一下，并把cout改printf结果就过了。

## 4）素数筛

P3383 【模板】线性筛素数

#include<cstdio>

using namespace std;

const long long N = 100000010;

bool isprime[N];

int prime[N], x;

void oulashai(int n)

{

for (int i = 2; i <= n; i++)

{

if (!isprime[i]) prime[x++] = i;

for (int j = 0; j < x; j++)

{

if (i \* prime[j] > n) break;

isprime[i \* prime[j]] = true;

if (i % prime[j] == 0) break;

}

}

}

int main() {

int n, m, num[N];

scanf("%d %d", &n, &m);

oulashai(n);

for (int i = 0; i < m; i++)

{

scanf("%d", &num[i]);

printf("%d\n", prime[num[i] - 1]);

}

return 0;

}

思路：素数筛方法最常见3种，第一种暴力筛，两个for循环测试该数是不是素数，当然这个筛法十有八九会超时。再有就是埃氏筛，先把2的倍数筛掉，然后下一个没被筛掉的是3，让后把3的倍数筛掉，下一个就是5以此类推，这个筛法比暴力筛好点，但最大的不足就是重复筛选，比如6被2筛了以此，又会被3筛一次，这就有点多次一举。于是帅气的欧拉筛就出来了（本题也是用欧拉筛来做的）。欧拉筛在埃氏筛的基础上多加了些条件使得筛数的过程中不会重复筛选。具体筛法就是设个数组判断该数是否为素数，一个数组用来存素数。每个数字都会进行筛选，当该数能被当前素数整除时结束（超出计算范围和素数的存储范围也结束）。

## 5）乘法逆元

P3811 【模板】乘法逆元

#include<iostream>

#include<stdio.h>

using namespace std;

int x, y;

void exgcd(int a, int b, int& x, int& y)

{

if (b == 0)

{

x = 1;

y = 0;

return;

}

exgcd(b, a % b, x, y);

int temp = x;

x = y;

y = temp - a / b \* y;

return;

}

int main()

{

register int n, p, i;

cin >> n >> p;

for (i = 1; i <= n; i++)

{

exgcd(i, p, x, y);

printf("%d\n", (x % p + p) % p);

}

return 0;

}

思路：乘法逆元的算法很多，由于上面写过了拓展欧几里得定理，而乘法逆元又可以用拓展欧几里得定理来做，所以我尝试了一下用拓展欧几里得来做。首先拓展欧几里得的算法就不多说了，主要是怎么用其实拓展欧几里得都要联系上同余定理，设我们要算的数是x要求i的逆元，mod的数是p那么ix恒等于1(mod p)就可以写成ix+py=1；因为i和p互质所以1也是他们的gcd于是解出x就好。(最后一个点超时没有过)

## 6）费马小定理

P3811 【模板】乘法逆元

#include<iostream>

#include<stdio.h>

using namespace std;

#define ll long long

int x, y;

ll fpm(ll x, ll mod) {

ll p = mod - 2;

ll ans = 1;

x %= mod;

while (p > 0)

{

if (p % 2 != 0)

ans = ans \* x % mod;

p /= 2;

x = x \* x % mod;

}

return ans;

}

int main()

{

int n, p;

cin >> n >> p;

for (int i = 1; i <= n; i++)

{

ll x = fpm(i, p);

printf("%lld\n", x);

}

return 0;

}

思路：依旧是那道题，乘法逆元其实还可以用费马小定理来做，费马小定理大概如下：假如p是质数，且gcd(a,p)=1，那么 a^(p-1)≡1（mod p） 两边都mod p；即：假如a是整数，p是质数，且a,p互质(即两者只有一个公约数1)，那么a的(p-1)次方除以p的余数恒等于1。这里设逆元为n则可以列出式子nx≡1（mod p），又因为费马小定理所以a^(p−1)≡1(mod p)因此x≡a^(p−2)(mod p)这是逆元问题就转化为了一个快速幂问题，具体做法参照上面就好了。（最后两个点超时，这道题真让我怀疑人生）

## 强化练习

### 1) HDU- 2161 Primes

#include<iostream>

#include<cstring>

using namespace std;

const int N = 50000;

int isprime[N], x = 0;

int prime[N];

void oulashai(int n)

{

memset(isprime, 0, sizeof(isprime));

for (int i = 2; i <= n; i++)

{

if (!isprime[i]) prime[x++] = i;

for (int j = 0; j < x; j++)

{

if (i \* prime[j] > n) break;

isprime[i \* prime[j]] = true;

if (i % prime[j] == 0) break;

}

}

}

int main()

{

int n, i = 1;

oulashai(N);

while (cin >> n)

{

if (n <= 0)break;

if (n == 1 || n == 2)

{

cout << i << ": no" << endl;

i++;

}

else if (isprime[n])

{

cout << i << ": no" << endl;

i++;

}

else

{

cout << i << ": yes" << endl;

i++;

}

}

return 0;

}

思路：这是一道素数筛的题，把素数筛出来然后特殊处理1，2，0然后，每输入一个数字判断是否素数，输出结果。

### 2) POJ - 2478 Farey Sequence

#include <iostream>

using namespace std;

const long long N = 3000001;

long long cprime = 0, prime[N], phi[N], n, i, j;

bool isprime[N];

void oula() {

phi[1] = 1;

for (i = 2; i <= N; i++)

{

if (!isprime[i])

{

prime[++cprime] = i;

phi[i] = i - 1;

}

for (j = 1; j <= cprime && i \* prime[j] <= N; j++)

{

isprime[i \* prime[j]] = true;

if (i % prime[j] == 0)

{

phi[i \* prime[j]] = phi[i] \* prime[j];

break;

}

else

{

phi[i \* prime[j]] = phi[i] \* (prime[j] - 1);

}

}

}

}

int main() {

oula();

long long n;

long long sum = 0;

while (cin >> n)

{

if (n == 0)

continue;

sum = 0;

for (long long i = 2; i <= n; i++)

{

sum += phi[i];

}

cout << sum << endl;

}

return 0;

}

思路：这道题与要考的是欧拉函数，欧拉函数的主要性质有：

1.若a为质数,phi[a]=a-1;

2.若a为质数,b mod a=0,phi[a\*b]=phi[b]\*a

3.若a,b互质,phi[a\*b]=phi[a]\*phi[b]

利用这几个性质可以写出欧拉函数。然后算出前n个数的欧拉函数值之和，输出就好了。

### 3) POJ – 1284 Primitive Roots

#include <iostream>

#include <cstring>

using namespace std;

const int N = 1e5 + 10;

int cprime = 0, prime[N], phi[N], n;

bool isprime[N];

void oula() {

phi[1] = 1;

for (int i = 2; i <= N; i++)

{

if (!isprime[i])

{

prime[++cprime] = i;

phi[i] = i - 1;

}

for (int j = 1; j <= cprime && i \* prime[j] <= N; j++)

{

isprime[i \* prime[j]] = true;

if (i % prime[j] == 0)

{

phi[i \* prime[j]] = phi[i] \* prime[j];

break;

}

else

{

phi[i \* prime[j]] = phi[i] \* (prime[j] - 1);

}

}

}

}

int main() {

oula();

int n;

while (cin >> n) {

cout << phi[n - 1] << endl;

}

return 0;

}

思路：这道题考也是欧拉函数，具体欧拉函数的写法上面写过了，就不多说了。

## 专题总结：

这一专题学的是数论，而数论主要是要学如何运用一些数学知识来解一些题。主要难点在于运用，有时看见一道题都不知道要用那种知识（除非是很明显那种）。

# 3.字符串

## 1）KMP

#include<iostream>

#include<string.h>

#include<stdlib.h>

using namespace std;

const int N = 1000000;

void prefixTable(char pattern[], int prefix[])

{

prefix[0] = 0;

int len = 0, i = 1;

int n = strlen(pattern);

while (i < n)

{

if (pattern[i] == pattern[len])

{

len++;

prefix[i] = len;

i++;

}

else

{

if (len > 0)

{

len = prefix[len - 1];

}

else

{

prefix[i] = len;

i++;

}

}

}

}

void change(int prefix[], int n)

{

for (int i = n - 1; i > 0; i--)

{

prefix[i] = prefix[i - 1];

}

prefix[0] = -1;

}

void kmp(char text[], char pattern[])

{

int n = strlen(pattern);

int m = strlen(text);

int\* prefix = (int\*)malloc(sizeof(int) \* n);

int i = 0, j = 0;

prefixTable(pattern, prefix);

change(prefix, n);

while (i < m)

{

if (j == n - 1 && text[i] == pattern[j])

{

cout << i - j + 1 << endl;

j = prefix[j];

}

if (text[i] == pattern[j])

{

i++;

j++;

}

else

{

j = prefix[j];

if (j == -1)

{

i++;

j++;

}

}

}

}

int main() {

char pattern[N], text[N];

int list[N];

cin >> text;

cin >> pattern;

for (int i = 0; i < strlen(pattern); i++)

list[i] = pattern[i];

prefixTable(pattern, list);

kmp(text, pattern);

for (int i = 0; i < strlen(pattern); i++)

cout << list[i] << " ";

return 0;

}

思路：kmp算法与普通匹配子串的方法来的快。普通方法的时间复杂度是O(m\*n)而kmp算法可以将时间复杂度降到O(m+n)，算法的主要思路就是先创建一个数组用来存储该字符串（要匹配的子串）的最长前后缀的长度，然后利用这个数组来用进行字符串匹配。先讲讲如何获得这个数组。首先写一个函数，把第一个字符的值设为0，然后在该字符串长度内，看当前字符与目标字符是否相同，相同则长度+1把这个值给当前的数组，否则长度设为以当前长度-1为下标的数组的值，继续匹配直到长度为0（如果前面的匹配都没有相同的字符），则把0赋给当前位置，并跳到下一部分来匹配。得到这个数组以后，把数组整体往后移，把第一个设为-1.就开始匹配了。匹配的方法是，先创建两个整形变量i，j记录子串和要匹配的字符串的下标（暂且称其为指针，当然不是传统意义上那种）。i指向当前被匹配的字符串的下标，j指向匹配子串的下标。当i指向的字符与j指向的字符相同则把指针往后移，当整个子串都匹配（即j为n，我这里写成j=n-1&&两字符相等，感觉这样快点）。如果中途匹配不成功j=下标为j的数组的值。如果j==-1即已经搜到了第一个字符，也就是说第一个字符也不匹配，则i和j都往后移一位。

## 2）Manacher

**经典例题：P3805 【模板】manacher算法**

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

const int N = 32000005;

char str[N], newstr[N];

int len, newlen, p[N], ans;

void init()

{

newstr[0] = '$';

newstr[1] = '#';

for (int i = 0; i < len; i++)

{

newstr[i \* 2 + 2] = str[i];

newstr[i \* 2 + 3] = '#';

}

newlen = len \* 2 + 2;

newstr[newlen] = '\*';

}

void manacher()

{

int id = 0, mx = 0;

for (int i = 1; i < newlen; i++)

{

if (mx > i)

p[i] = min(p[2 \* id - i], mx - i);

else

p[i] = 1;

while (newstr[i + p[i]] == newstr[i - p[i]])

p[i]++;

if (p[i] + i > mx)

{

mx = p[i] + i;

id = i;

}

}

}

int main() {

cin >> str;

len = strlen(str);

init();

manacher();

ans = 0;

for (int i = 0; i < newlen; i++)

ans = max(ans, p[i]);

cout << ans - 1 << endl;

}

思路：manacher（马拉车）算法是用来解决最大回文串问题的，原本在不用马拉车算法的情况下，一般搜过去时间复杂度是O(n^2)，而使用马拉车算法的话可以把时间复杂度讲到O(n)。马拉车算法的主要思路是这样子的。首先要处理当前字符串，由于回文有两种，一种是奇数的一种是偶数的。正常情况下要分开处理，但是这里有一个巧妙的方法，就是上面写的init函数。其方法大概是在头尾加一个不常用的字符然后在每个字符之间插入一个‘#’。这样子无论是什么串都好最终都能处理成一个奇数回文串。接下来用一个mx变量记录最右边界，当当前i（即当前的串中心下标）>i时，又有两种情况，一种以其对称的字符为中心的最长字符小于外边界，那么该字符的最长回文串长度等于其对称的回文串长度，否则如果超出外边界，则外边界的部分要重新匹配。当i超出外边界时，也是重新匹配。每次处理完后看该字符为中心的最大字符串的最右边是否超出边界，若超出更新id和外边界mx。以此已知匹配下去。

## 强化练习：

### 1）HDU – 1711 Number Sequence

#include<iostream>

#include<string.h>

#include<stdlib.h>

using namespace std;

const int N = 1000000;

int flag;

int pattern[N], prefix[N], text[N], num, tlen, plen;

void prefixTable(int pattern[], int prefix[], int n)

{

prefix[0] = 0;

int len = 0, i = 1;

while (i < n)

{

if (pattern[i] == pattern[len])

{

len++;

prefix[i] = len;

i++;

}

else

{

if (len > 0)

{

len = prefix[len - 1];

}

else

{

prefix[i] = len;

i++;

}

}

}

}

void change(int prefix[], int n)

{

for (int i = n - 1; i > 0; i--)

{

prefix[i] = prefix[i - 1];

}

prefix[0] = -1;

}

void kmp(int text[], int pattern[], int n, int m)

{

int i = 0, j = 0;

prefixTable(pattern, prefix, n);

change(prefix, n);

while (i < m)

{

if (j == n - 1 && text[i] == pattern[j])

{

if (flag == 0)

{

cout << i - j + 1 << endl;

flag = 1;

}

j = prefix[j];

}

if (text[i] == pattern[j])

{

i++;

j++;

}

else

{

int temp = j;

j = prefix[temp];

if (j == -1)

{

i++;

j++;

}

}

}

}

int main() {

cin >> num;

for (int i = 0; i < num; i++)

{

cin >> tlen >> plen;

flag = 0;

memset(pattern, 0, sizeof(pattern));

memset(text, 0, sizeof(text));

for (int j = 0; j < tlen; j++)

cin >> text[j];

for (int j = 0; j < plen; j++)

cin >> pattern[j];

kmp(text, pattern, plen, tlen);

if (flag == 0)cout << -1 << endl;

}

return 0;

}

思路：和传统kmp做法一样，只是一些其他细节有点改变，如这次的数组的长度是给定的，所以不用获取字符串长度，而且数组使用int。其他和传统kmp都差不都。匹配到的第一次输出其位置。并标记为已找到，若找不到则最后输出-1。

### 2）HDU – 2087 剪花布条

#include<iostream>

#include<string.h>

#include<stdlib.h>

using namespace std;

const int N = 1000000;

char pattern[N], text[N];

int ans;

void prefixTable(char pattern[], int prefix[])

{

prefix[0] = 0;

int len = 0, i = 1;

int n = strlen(pattern);

while (i < n)

{

if (pattern[i] == pattern[len])

{

len++;

prefix[i] = len;

i++;

}

else

{

if (len > 0)

{

len = prefix[len - 1];

}

else

{

prefix[i] = len;

i++;

}

}

}

}

void change(int prefix[], int n)

{

for (int i = n - 1; i > 0; i--)

{

prefix[i] = prefix[i - 1];

}

prefix[0] = -1;

}

void kmp()

{

int n = strlen(pattern);

int m = strlen(text);

int\* prefix = (int\*)malloc(sizeof(int) \* n);

int i = 0, j = 0;

prefixTable(pattern, prefix);

change(prefix, n);

while (i < m)

{

if (n == 1)

{

if (text[i] == pattern[j])

{

ans++;

j = 0;

i++;

}

else

{

i++;

}

}

else

{

if (j == n - 1 && text[i] == pattern[j])

{

ans++;

j = 0;

i++;

}

if (text[i] == pattern[j])

{

i++;

j++;

}

else

{

j = prefix[j];

if (j == -1)

{

i++;

j++;

}

}

}

}

}

int main() {

cin >> text;

if (text[0] == '#')break;

cin >> pattern;

ans = 0;

kmp();

cout << ans << endl;

return 0;

}

思路：这道题是kmp的一个小变形，不像以往以往跳出的操作有点不一样搜完一个子串后，直接往后搜而不是重复搜。每次的到匹配后ans++，其他都和之前的kmp差不多。

### 3）HUD 3068 最长回文

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

const int N = 32000005;

char str[N], newstr[N];

int len, newlen, p[N], ans;

void init()

{

newstr[0] = '$';

newstr[1] = '#';

for (int i = 0; i < len; i++)

{

newstr[i \* 2 + 2] = str[i];

newstr[i \* 2 + 3] = '#';

}

newlen = len \* 2 + 2;

newstr[newlen] = '\*';

}

void manacher()

{

int id = 0, mx = 0;

for (int i = 1; i < newlen; i++)

{

if (mx > i)

p[i] = min(p[2 \* id - i], mx - i);

else

p[i] = 1;

while (newstr[i + p[i]] == newstr[i - p[i]])

p[i]++;

if (p[i] + i > mx)

{

mx = p[i] + i;

id = i;

}

}

}

int main() {

while (cin >> str)

{

len = strlen(str);

init();

manacher();

ans = 0;

for (int i = 0; i < newlen; i++)

ans = max(ans, p[i]);

cout << ans - 1 << endl;

}

}

思路：这道是典型的最大回文串问题，按照最大回文串模板来写就好了，主要是注意数组要开多大。

## 专题总结：

字符串的匹配入门的时候真的心态爆炸，刚开始学kmp和马拉车的时候真的好艰难。后来看了很多资料，对字符串匹配思想的理解渐渐加深。就自己的见解，这几个字符串的算法的目的就是把各种暴力操作（复杂度较高）弄成一个线性时间的操作。然后沿着这个思路就能更好的理解模板代码的写法。

# 4.动态规划

## 1）背包dp

经典题：P1048 采药

#include<iostream>

#include<algorithm>

using namespace std;

const int N = 1005;

int w[N], v[N], dp[N][N];

int main()

{

int t, m;

cin >> t >> m;

for (int i = 1; i <= m; i++)

cin >> w[i] >> v[i];

for (int i = 1; i <= m; i++)

for (int j = t; j >= 0; j--)

{

if (w[i] <= j)

dp[i][j] = max(dp[i - 1][j - w[i]] + v[i], dp[i - 1][j]);

else

dp[i][j] = dp[i - 1][j];

}

cout << dp[m][t] << endl;

return 0;

}

思路：背包dp主要解决一定空间中拿物品怎么拿才能拿到最大值的问题。通常我们可以用一个二维数组来做，横轴代表背包容量，而竖轴代表第几件物品。然后第一个循环枚举物品第二个循环枚举容量，在规定范围内dp[i][j] = max(dp[i - 1][j - w[i]] + v[i], dp[i - 1][j])。即比较取这个物品与不取这个物品看那种做法更优，最后输出答案。

## 2）区间dp

经典题：51Nod – 1021 石子归并

#include <iostream>

#include<cstring>

#include<algorithm>

using namespace std;

const int N = 101;

const int M = 1000000;

int stone[N], dp[N][N], sum[N];

int main()

{

int n;

cin >> n;

memset(sum, 0, sizeof(sum));

memset(dp, M, sizeof(dp));

for (int i = 1; i <= n; i++) {

cin >> stone[i];

sum[i] = sum[i - 1] + stone[i];

dp[i][i] = 0;

}

for (int len = 1; len <= n; len++)

{

for (int j = 1; j + len <= n + 1; j++)

{

int end = j + len - 1;

for (int i = j; i < end; i++)

{

dp[j][end] = min(dp[j][end], dp[j][i] + dp[i + 1][end] + sum[end] - sum[j - 1]);

}

}

}

cout << dp[1][n] << endl;

return 0;

}

思路：区间dp的大概套路是三个for循环，先枚举长度，即每次合并的长度，接下来枚举起点，即每次枚举的起点，最后枚举分割点。创建一个dp数组用来枚举i到j的最优解，最终输出dp[1][n]（即从起点到终点的最优解）以这道题为例首先输入石头的堆数，然后分别输入每堆石头的数量。同时用sum数组来存储前n项和。接着三个for循环枚举，每次更新最优解（即每段距离之间最小的解）。最后输出dp[1][n]。

## 强化练习

### 1）P1049 装箱问题

#include<iostream>

#include<algorithm>

using namespace std;

const int N = 20001;

int m, n, f[N], w[N];

int main() {

int i, j;

cin >> m >> n;

for (i = 1; i <= n; i++)

cin >> w[i];

for (i = 1; i <= n; i++)

for (j = m; j >= w[i]; j--)

f[j] = max(f[j], f[j - w[i]] + w[i]);

cout << m - f[m] << endl;

}

思路：这是一个特殊的背包dp这个题中物品的重量即使物品的价值，然后求最优解（即最大能装多少）。最后输出剩余量就好了。

### 2)P1359 租用游艇

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

const int N = 205;

const int M = 1000000;

int r[N][N], dp[N], n;

int main()

{

memset(dp, M, sizeof(dp));

cin >> n;

dp[n] = 0;

for (int i = 1; i < n; i++)

for (int j = i + 1; j <= n; j++)

cin >> r[i][j];

for (int i = n - 1; i > 0; i--)

for (int j = i + 1; j <= n; j++)

dp[i] = min(dp[i], r[i][j] + dp[j]);

cout << dp[1] << endl;

return 0;

}

思路：先记录下左右点到另一个点的租金，然后开始dp从后往前（当然从前往后也可以），记录最后点到前面点的最佳方案，直到记录到起点位置，然后输出起点到终点的最佳方案。

### 3) P1060 开心的金明

#include<iostream>

#include<algorithm>

using namespace std;

const int N = 1005;

int w[N], v[N], dp[N][N];

int main()

{

int n, m, temp;

cin >> n >> m;

for (int i = 1; i <= m; i++)

{

cin >> w[i] >> temp;

v[i] = temp \* w[i];

}

for (int i = 1; i <= m; i++)

for (int j = n; j >= 0; j--)

{

if (w[i] <= j)

dp[i][j] = max(dp[i - 1][j - w[i]] + v[i], dp[i - 1][j]);

else

dp[i][j] = dp[i - 1][j];

}

cout << dp[m][n] << endl;

return 0;

}

思路：这是一个背包的变形，主要是要在数据输入时要进行一些处理，每次都要把权重乘以价值才能得到真正的价值，其他的就和平常的背包差不多了。

## 专题总结：

动态规划算法是通过拆分问题，定义问题状态和状态之间的关系，使得问题能够以递推（或者说分治）的方式去解决。目前就学了背包和区间dp而且刷的基本上都是模板题。所以比较水，以后还要慢慢练（听说dp进阶超难）。

# 5.数据结构

## 1）并查集

**经典例题：P3367 【模板】并查集**

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

const int N = 200001;

int n, m, z[N], x[N], y[N], parent[N], r[N];

int findRoot(int x)

{

int xRoot = x;

while (parent[xRoot] != -1)

xRoot = parent[xRoot];

return xRoot;

}

void unionNode(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot != yRoot)

{

if (r[xRoot] > r[yRoot])

parent[yRoot] = xRoot;

else if (r[xRoot] < r[yRoot])

parent[xRoot] = yRoot;

else

{

parent[xRoot] = yRoot;

r[yRoot]++;

}

}

}

bool inSameGroup(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot == yRoot)

return true;

else

return false;

}

int main() {

memset(parent, -1, sizeof(parent));

memset(r, 0, sizeof(r));

cin >> n >> m;

for (int i = 0; i < m; i++)

cin >> z[i] >> x[i] >> y[i];

for (int i = 0; i < m; i++)

{

if (z[i] == 1)

unionNode(x[i], y[i]);

else

{

if (inSameGroup(x[i], y[i]))

cout << "Y" << endl;

else

cout << "N" << endl;

}

}

}

思路：并查集是一种树型的数据结构，用于处理一些不相交集合（Disjoint Sets）的合并及查询问题。常常在使用中以森林来表示。（百度总结）在我看来就是用来解决点与点之间是否连通以及如何连通的方法（在下的粗陋见解）。主要操作有几部，首先用一个parent数组来存储某个数的父结点，当父结点为-1时，这个点就是这一组的根结点。当需要连接两个（组）结点时，只需找到两个点（或一组中的某个点）的根结点，然后把两个根结点连接起来。这里有个技巧每次连接时我们因该把树的高度小的部分接到大的部分，这样子树的高度就不会太高，导致搜根结点时时间太长。（我一开始就是因为这个原因超时的）。而判断是否连接就很简单了，只要两点的根结点相同就是连接的，否则就是不连接的。再讲讲这道题，首先看z[i]是1还是2，1则连接两个结点，2则判断两个结点是否连通。具体的操作就和上面说的一样就好了。

## 2）堆

**经典例题：P3378 【模板】堆**

#include<iostream>

#include<queue>

#include<cstdio>

using namespace std;

int n, op, num;

priority\_queue <int>q;

int main() {

scanf("%d", &n);

for (int i = 0; i < n; i++)

{

scanf("%d", &op);

if (1 == op)

{

scanf("%d", &num);

q.push(-num);

}

else if (2 == op)

cout << -q.top() << endl;

else

q.pop();

}

return 0;

}

思路：堆是一种特殊的二叉树，具备以下两种性质：1）每个节点的值都大于（或者都小于，称为最小堆）其子节点的值；2)树是完全平衡的，并且最后一层的树叶都在最左边。堆分大根和小根。大根堆的根结点是队列中数最大的，而小根堆的根结点是队列中数最小的。这道题就是用小根堆做的。首先用stl中的priority\_queue创建一个队列。priority\_queue原本是大根堆所以我们可以把每个数字的相反数压入队列元素的绝对值中，根结点是最小的，输出时，输出回它的相反值就好了。

## 强化训练

### HUD1232 畅通工程

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

const int N = 200001;

int n, m, x[N], y[N], parent[N], r[N], ans;

int findRoot(int x)

{

int xRoot = x;

while (parent[xRoot] != -1)

xRoot = parent[xRoot];

return xRoot;

}

void unionNode(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot != yRoot)

{

if (r[xRoot] > r[yRoot])

parent[yRoot] = xRoot;

else if (r[xRoot] < r[yRoot])

parent[xRoot] = yRoot;

else

{

parent[xRoot] = yRoot;

r[yRoot]++;

}

ans++;

}

}

bool inSameGroup(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot == yRoot)

return true;

else

return false;

}

int main() {

while (cin >> n >> m)

{

if (n == 0)break;

ans = 1;

memset(parent, -1, sizeof(parent));

memset(r, 0, sizeof(r));

for (int i = 0; i < m; i++)

cin >> x[i] >> y[i];

for (int i = 0; i < m; i++)

{

unionNode(x[i], y[i]);

}

cout << n - ans << endl;

}

}

思路：这是一道可以用并查集的题，主要是连通性的问题。每次连通数就加1。最后输出地点数减去搭桥数就是所需的答案。

### 2）HUD1213 How Many Tables

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

const int N = 200001;

int n, m, x[N], y[N], parent[N], r[N], ans, ca;

int findRoot(int x)

{

int xRoot = x;

while (parent[xRoot] != -1)

xRoot = parent[xRoot];

return xRoot;

}

void unionNode(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot != yRoot)

{

if (r[xRoot] > r[yRoot])

parent[yRoot] = xRoot;

else if (r[xRoot] < r[yRoot])

parent[xRoot] = yRoot;

else

{

parent[xRoot] = yRoot;

r[yRoot]++;

}

}

}

int main() {

cin >> ca;

for (int i = 0; i < ca; i++)

{

cin >> n >> m;

ans = 0;

memset(parent, -1, sizeof(parent));

memset(r, 0, sizeof(r));

for (int i = 0; i < m; i++)

{

cin >> x[i] >> y[i];

unionNode(x[i], y[i]);

}

for (int i = 1; i <= n; i++)

if (parent[i] == -1)

ans++;

cout << ans << endl;

}

}

思路：还是原来的模板只要做些小调整就好了。这道题先按他的要求接受测试例子数，然后输入人数和关系数，进行合并后遍历parent数组如果父结点是-1，ans++最后输出答案。

### 3）P1090 合并果子 / [USACO06NOV] Fence Repair G

#include<iostream>

#include<queue>

using namespace std;

int n, num, a, b, sum;

priority\_queue <int>q;

int main() {

cin >> n;

for (int i = 0; i < n; i++)

{

cin >> num;

q.push(-num);

}

for (int i = 0; i < n-1; i++)

{

a = q.top();

q.pop();

b = q.top();

q.pop();

sum += -a - b;

q.push(a + b);

}

cout << sum << endl;

return 0;

}

思路：这道题可以用堆来做，思路就是使用小根堆，先把所有元素压入堆中，然后每次弹出两个最小的元素把他们合并后重新压入堆中，同时把这连个元素的和加入sum中。最后输出sum。

## 专题总结：

数据结构是个神奇的专题，我感觉这个专题打开了一个新世界，这算是我第一次开始接触C++的stl，然后发现stl特别香，像是vector, queue, priority\_queue,map等等真的发现特别好用。但是stl用的实在不熟练，以后还得好好练，至于线段树，晚点我会好好学的。

# 6. 图论

## 1）最小生成树

**经典例题：P3366 【模板】最小生成树**

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

#define ll long long

const int N = 200001;

int n, m, parent[N], r[N];

ll ans = 0;

struct edge

{

int s, e;

ll cost;

}edges[N];

int findRoot(int x)

{

int xRoot = x;

while (parent[xRoot] != -1)

xRoot = parent[xRoot];

return xRoot;

}

void unionNode(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot != yRoot)

{

if (r[xRoot] > r[yRoot])

parent[yRoot] = xRoot;

else if (r[xRoot] < r[yRoot])

parent[xRoot] = yRoot;

else

{

parent[xRoot] = yRoot;

r[yRoot]++;

}

}

}

bool inSameGroup(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot == yRoot)

return true;

else

return false;

}

bool cmp(edge a, edge b)

{

return a.cost < b.cost;

}

void kruskal()

{

std::sort(edges + 1, edges + 1 + m, cmp);

for (int i = 1; i <= m; i++)

{

if (inSameGroup(edges[i].s, edges[i].e))

continue;

unionNode(edges[i].s, edges[i].e);

ans += edges[i].cost;

}

}

int main()

{

memset(parent, -1, sizeof(parent));

memset(r, 0, sizeof(r));

cin >> n >> m;

for (int i = 1; i <= m; i++)

cin >> edges[i].s >> edges[i].e >> edges[i].cost;

kruskal();

for (int i = 1; i <= n; i++)

if (!inSameGroup(i, 1))

ans = -1;

if (-1 == ans)

cout << "orz" << endl;

else

cout << ans << endl;

return 0;

}

思路：最小生成数kruskal算法的基本思路是先用一个结构体记录每条边的起点，终点，以及边的权重。以其权重进行排序，小的在前，大的在后。每次取权重最小的那个边如果那个边所连接的连个点是连通的则舍去这条边。如果不连通则连通这两个点，并ans+=这个边的权重。最后判断起点与其他点是否连通，如果都连通则输出答案。如果不连通则说明该图没有完全连通。至于是否连通等问题，用并查集来做就好了。

## 2）最短路径

**经典例题：P4779 【模板】单源最短路径（标准版）**

#include<iostream>

#include<cstring>

#include<queue>

using namespace std;

#define MAX 0x7fffffff

const int N = 100010, M = 500010;

int head[N], dis[N], cnt;

bool vis[N];

int n, m, s;

struct edge

{

int to, dis, next;

}e[M];

void add\_edge(int u, int v, int d)

{

cnt++;

e[cnt].dis = d;

e[cnt].to = v;

e[cnt].next = head[u];

head[u] = cnt;

}

struct node

{

int dis;

int pos;

bool operator <(const node& x)const

{

return x.dis < dis;

}

};

priority\_queue<node> q;

void dijkstra()

{

dis[s] = 0;

q.push({ 0, s });

while (!q.empty())

{

node tmp = q.top();

q.pop();

int x = tmp.pos, d = tmp.dis;

if (!vis[x])

{

vis[x] = 1;

for (int i = head[x]; i; i = e[i].next)

{

int y = e[i].to;

if (dis[y] > dis[x] + e[i].dis && !vis[y])

{

dis[y] = dis[x] + e[i].dis;

q.push({ dis[y], y });

}

}

}

}

}

int main()

{

cin >> n >> m >> s;

for (int i = 1; i <= n; i++)dis[i] = MAX;

for (int i = 0; i < m; i++)

{

int u, v, d;

cin >> u >> v >> d;

add\_edge(u, v, d);

}

dijkstra();

for (int i = 1; i <= n; i++)

cout << dis[i] << " ";

return 0;

}

思路：这是道最短路径问题，最短路径问题可以用dijkstra算法来求。dijkstra的基本思路如下：首先创建一个数组用来存储起点到各个点的距离，首先把这个数组初始化为一个很大的数。同时记下起点到自己的距离为0。从起点开始，更新所有的邻边的距离值，然后找到距离值最小的那个点，接下来看看该点的所有邻边，并把这几个的距离值更新，更过的就不用更了以此类推，最总推出所有的最短路径。

## 强化训练

### HUD 1863 畅通工程

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

#define ll long long

const int N = 200001;

int n, m, parent[N], r[N];

ll ans = 0;

struct edge

{

int s, e;

ll cost;

}edges[N];

int findRoot(int x)

{

int xRoot = x;

while (parent[xRoot] != -1)

xRoot = parent[xRoot];

return xRoot;

}

void unionNode(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot != yRoot)

{

if (r[xRoot] > r[yRoot])

parent[yRoot] = xRoot;

else if (r[xRoot] < r[yRoot])

parent[xRoot] = yRoot;

else

{

parent[xRoot] = yRoot;

r[yRoot]++;

}

}

}

bool inSameGroup(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot == yRoot)

return true;

else

return false;

}

bool cmp(edge a, edge b)

{

return a.cost < b.cost;

}

void kruskal()

{

std::sort(edges + 1, edges + 1 + m, cmp);

for (int i = 1; i <= m; i++)

{

if (inSameGroup(edges[i].s, edges[i].e))

continue;

unionNode(edges[i].s, edges[i].e);

ans += edges[i].cost;

}

}

int main()

{

while (cin >> m >> n) {

ans = 0;

memset(parent, -1, sizeof(parent));

memset(r, 0, sizeof(r));

if (0 == m)break;

for (int i = 1; i <= m; i++)

cin >> edges[i].s >> edges[i].e >> edges[i].cost;

kruskal();

for (int i = 1; i <= n; i++)

if (!inSameGroup(i, 1))

ans = -1;

if (-1 == ans)

cout << "?" << endl;

else

cout << ans << endl;

}

return 0;

}

思路：这道算是典型的最小生成树问题了，这道题直接套kruskal和并查集模板写就好了。注意数据处理，每次都要初始化。

### 2）POJ – 1287 Networking

#include<iostream>

#include<algorithm>

#include<cstring>

using namespace std;

#define ll long long

const int N = 200001;

int n, m, parent[N], r[N];

ll ans = 0;

struct edge

{

int s, e;

ll cost;

}edges[N];

int findRoot(int x)

{

int xRoot = x;

while (parent[xRoot] != -1)

xRoot = parent[xRoot];

return xRoot;

}

void unionNode(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot != yRoot)

{

if (r[xRoot] > r[yRoot])

parent[yRoot] = xRoot;

else if (r[xRoot] < r[yRoot])

parent[xRoot] = yRoot;

else

{

parent[xRoot] = yRoot;

r[yRoot]++;

}

}

}

bool inSameGroup(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot == yRoot)

return true;

else

return false;

}

bool cmp(edge a, edge b)

{

return a.cost < b.cost;

}

void kruskal()

{

sort(edges + 1, edges + 1 + n, cmp);

for (int i = 1; i <= n; i++)

{

if (inSameGroup(edges[i].s, edges[i].e))

continue;

unionNode(edges[i].s, edges[i].e);

ans += edges[i].cost;

}

}

int main()

{

while (cin >> m) {

if (0 == m)break;

cin >> n;

ans = 0;

memset(parent, -1, sizeof(parent));

memset(r, 0, sizeof(r));

if (0 == n)

cout << "0" << endl;

else

{

for (int i = 1; i <= n; i++)

cin >> edges[i].s >> edges[i].e >> edges[i].cost;

kruskal();

cout << ans << endl;

}

}

return 0;

}

思路：这也是道简单的最小生成树问题，一开始我把问题想复杂了，因为输入会有重复，及两个点之间的权值有多个，一开始我的思路是把每个点的值先保存下来，每次输入时判断当前两点间的权值是否为最小，不是则更新，是不变。后来仔细想想自己就是个憨憨。kruskal算法开场就把最小权值的边往前排了，所以已经保证了取的边一定是当前两点间权值的最小值了。我还想那么多干嘛[哭笑]。恍然大悟后，就套模板去了。

### 3）HUD 1875 畅通工程再续（超时）

#include<iostream>

#include<algorithm>

#include<cstring>

#include<cmath>

using namespace std;

const int N = 1001;

int n, parent[N], r[N], k, num;

double ans = 0;

struct node

{

int x, y;

}nodes[N];

struct edge

{

int s, e;

double cost;

}edges[N];

int findRoot(int x)

{

int xRoot = x;

while (parent[xRoot] != -1)

xRoot = parent[xRoot];

return xRoot;

}

void unionNode(int x, int y, int i)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot != yRoot && edges[i].cost >= 10 && edges[i].cost <= 1000)

{

if (r[xRoot] > r[yRoot])

parent[yRoot] = xRoot;

else if (r[xRoot] < r[yRoot])

parent[xRoot] = yRoot;

else

{

parent[xRoot] = yRoot;

r[yRoot]++;

}

num++;

}

}

bool inSameGroup(int x, int y)

{

int xRoot = findRoot(x);

int yRoot = findRoot(y);

if (xRoot == yRoot)

return true;

else

return false;

}

bool cmp(edge a, edge b)

{

return a.cost < b.cost;

}

int main()

{

int cases;

scanf("%d", &cases);

for (int ca = 0; ca < cases; ca++)

{

scanf("%d", &n);

ans = 0;

k = 1;

num = 0;

memset(parent, -1, sizeof(parent));

memset(r, 0, sizeof(r));

for (int i = 1; i <= n; i++)

scanf("%d%d", &nodes[i].x, &nodes[i].y);

for (int i = 1; i < n; i++)

{

for (int j = i + 1; j <= n; j++)

{

edges[k].s = i;

edges[k].e = j;

edges[k++].cost = sqrt((nodes[i].x - nodes[j].x) \* (nodes[i].x - nodes[j].x) + (nodes[i].y - nodes[j].y) \* (nodes[i].y - nodes[j].y) \* 1.0);

}

}

k--;

sort(edges + 1, edges + 1 + k, cmp);

for (int i = 1; i <= k; i++)

{

if (inSameGroup(edges[i].s, edges[i].e))

continue;

unionNode(edges[i].s, edges[i].e, i);

ans += edges[i].cost;

}

if (num != n - 1)

printf("oh!\n");

else

printf("%.1f\n", ans \* 100);

}

return 0;

}

思路：这道题也是最小生成数问题，但是要加点变化，首先输入的是点，所以要用两个循环把点与点之间的距离算出来，并保存在edges中，转换完以后才能用kruskal，而且在连通时也要有条件，点坐标必须大于等于10，小于等于1000。否则就不连接。大概思路如此，但我超时过不了。（伤心）

### 4）HUD-畅通工程续(WA)

#include<iostream>

#include<cstring>

#include<queue>

#include<algorithm>

using namespace std;

#define MAX 0x7fffffff

const int N = 100010, M = 500010;

int head[N], dis[N], cnt;

bool vis[N];

int n, m, s, t, a, b;

struct edge

{

int to, dis, next;

}e[M];

void add\_edge(int u, int v, int d)

{

e[++cnt].dis = d;

e[cnt].to = v;

e[cnt].next = head[u];

head[u] = cnt;

}

struct node

{

int dis;

int pos;

bool operator <(const node& x)const

{

return x.dis < dis;

}

};

priority\_queue<node>q;

void dijkstra()

{

dis[s] = 0;

q.push({ 0, s });

while (!q.empty())

{

node tmp = q.top();

q.pop();

int x = tmp.pos, d = tmp.dis;

if (!vis[x])

{

vis[x] = 1;

for (int i = head[x]; i; i = e[i].next)

{

int y = e[i].to;

if (dis[y] > dis[x] + e[i].dis && !vis[y])

{

dis[y] = dis[x] + e[i].dis;

q.push({ dis[y], y });

}

}

}

}

}

int main()

{

while (cin >> n >> m)

{

memset(head, 0, sizeof(head));

memset(dis, 0, sizeof(dis));

memset(vis, 0, sizeof(vis));

cnt = 0;

for (int i = 0; i < n; i++)dis[i] = MAX;

for (int i = 0; i < m; i++)

{

int u, v, d;

cin >> u >> v >> d;

add\_edge(u, v, d);

}

cin >> a >> b;

s = min(a, b);

t = max(a, b);

dijkstra();

if (dis[t] == MAX)

cout << -1 << endl;

else

cout << dis[t] << endl;

}

return 0;

}

思路：与杭电畅通工程死磕到底。这道题算是最短路径的题。我本来以为是一道直接可以套模板的题。套完模板以后wa。一开始以为是初始化的问题，处理了一下初始化，结果还是wa。测试多了几个实例，发现可能与边的方向问题，处理完以后，还是wa。然后就不知道哪里错。绝望……

## 专题总结：

这个专题主要是一些和图有关的题，多是一些点和边的操作。因为只学了一点，题也刷的比较少所以没什么太大的感觉。以后再好好刷。

# Codeforces

## Codeforces Round #617 (Div. 3)

### A）Array with Odd Sum

#include<iostream>

using namespace std;

const int N = 200001;

int n, cnt, temp, o, e;

int main()

{

cin >> n;

for (int i = 0; i < n; i++)

{

o = 0;

e = 0;

cin >> cnt;

for (int j = 0; j < cnt; j++)

{

cin >> temp;

if (temp % 2 == 0)

e++;

else

o++;

}

if (o == 0 && e > 0)

cout << "NO" << endl;

else if (e == 0 && o % 2 == 0)

cout << "NO" << endl;

else

cout << "YES" << endl;

}

return 0;

}

思路：这道题的思路是要判断是否经过一顿操作后，这个数组的和可以等于奇数可以则输出YES否则输出NO，我们来分析一下如何才能加出一个奇数。那就需要奇数个奇数相加，因此我们只要有机会在数组复制出奇数个奇数，就有机会凑出何为奇数的数组。那么我们分析一下什么情况下不可以呢？第一种，全是偶数那无论如何都弄不出一个奇数。第二种，只有偶数个奇数，这种情况下，是无法凑的。但凡数组中既有偶数也有奇数，无论数组的长度是奇数还是偶数，都可以把数组的奇数操作成奇数，剩下的全操作成偶数，那么就有机会凑出奇数和。

### B）Food Buying

#include<iostream>

using namespace std;

long long n, sum, num;

int main()

{

cin >> n;

for (int i = 0; i < n; i++)

{

cin >> num;

sum = 0;

while (num / 10 != 0)

{

sum += (num / 10) \* 10;

num = (num % 10) + (num / 10);

}

cout << sum + num << endl;

}

return 0;

}

思路：这道题大概思路就是每花10元就返回1元，所以每次我们就把当前的钱数的个位分离出来，然后把这些钱加入sum中。然后就可以拿到钱数/10的回扣，把之前的个位加回进去，然后再重复之前的操作。直到回扣加个位小于10为止，此时把剩下的钱加入sum中，就是我们所要的答案。

### C) Yet Another Walking Robot

#include<iostream>

#include<vector>

#include<map>

#define Node make\_pair(x, y)

using namespace std;

const int N = 200000;

int n, m;

char str[N];

int main()

{

cin >> m;

for (int i = 0; i < m; i++)

{

int x = 0, y = 0, len = 999999, ansx = 0, ansy = 0;

map<pair<int, int>, int>h;

h[make\_pair(0, 0)] = 0;

cin >> n >> str;

for (int j = 0; j < n; j++)

{

if (str[j] == 'L')x--;

else if (str[j] == 'R')x++;

else if (str[j] == 'U')y++;

else if (str[j] == 'D')y--;

if (h.count(Node))

{

if (len > j - h[Node])

{

ansx = h[Node] + 1;

ansy = j + 1;

len = j - h[Node];

}

}

h[Node] = j + 1;

}

if (len != 999999)

cout << ansx << " " << ansy << endl;

else

cout << "-1" << endl;

}

return 0;

}

思路：这道题的大致题意是找出移动路径中，最小的环，然后输出。这道题的做法是，用一个map，以每次的x，y作为key以这一步出现的字符数组下标为值。对不同的字符有不同的操作，操作后看看是否已经存在这个key如果存在则说明形成环了，此时看看这个环的长度，如果这个环比之前的环的长度要短，则更新答案，否则跳过，以此类推。

### D) Fight with Monsters

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

int n, a, b, k;

vector<int> v;

int main() {

cin >> n >> a >> b >> k;

int hp, ans = 0;

for (int i = 1; i <= n; i++) {

cin >> hp;

int lhp = (hp - 1) % (a + b) + 1;

if (lhp <= a)

ans++;

else

v.push\_back(lhp - a);

}

sort(v.begin(), v.end());

for (int i = 0; i < v.size(); i++) {

int lhp = (v[i] + a - 1) / a;

if (k >= lhp) {

k -= lhp;

ans++;

}

}

cout << ans << endl;

return 0;

}

思路：这道题的大致是两个打怪，总共有n个怪，每个怪都有hp点血，我攻击力为a，对手攻击力为b，我有个技能能让自己多打一个回合，技能能用k次。打怪的顺序是这样的我先a一下对手a一下。这道题首先看怪的hp一个回合，我和对手各a一下会a出（a+b）点血。我们的目的是由我来a死怪物，这样才能得分。所以这道题的基本思路是这样子的，首先用(hp - 1) % (a + b) + 1来算出最后一击，如果得出的lhp可由我收割，则ans++，否则把剩下的lhp放入v中，晚点再处理。等到所有的怪都处理一遍以后，由小到大排v中的怪的体力，剩下体力小的在前（体力剩的越少，所需要消耗的技能数就越少），然后一个个处理，需要多少次技能数，就是技能数减多少，同时ans++，最后输出答案。

### E1、E2（String Coloring）、F（Berland Beauty）

这几道真的不会，题目没太看懂的情况下，完全没思路，看题解也是一脸懵逼。

### 赛后总结：

这算是我第一次网不卡来做CF题了，看题懵逼，敲代码更懵逼，这次比较尴尬就算事后补题还有3补不出来，有点尴尬。之后再慢慢补。

## Codeforces Round #653 (Div. 3)

### A) Required Remainder

#include<iostream>

using namespace std;

int main()

{

int m;

cin >> m;

while (m--)

{

long long x, y, n, ans = 0;

cin >> x >> y >> n;

ans = n / x \* x + y;

if (ans > n)

ans -= x;

cout << ans << endl;

}

return 0;

}

思路：这道题主要是找出一个小于等于n的最大值，且这个值除以x后余数为y。这道题还是很好做的我们拿n/x\*x就可以得到一个能被x整除的小于n的最大值然后在把y加进去就可以了，放一手y加进去以后大于了n最后还要判断一下，超了就要减个x没超直接输出就好了。

### B) Multiply by 2, divide by 6

#include<iostream>

using namespace std;

int main()

{

int m;

cin >> m;

while (m--)

{

int n, t = 0;

cin >> n;

while (n % 6 == 0)

{

n /= 6;

t++;

}

while (n % 3 == 0 && n != 1)

{

n /= 3;

t += 2;

}

if (1 == n)

cout << t << endl;

else

cout << "-1" << endl;

}

return 0;

}

思路：这道题大致思路是总共两个操作\*2和/6最后能弄成1则输出步骤数，否则输出-1。这道题仔细分析一下，我们可以先把这个数除以6，每除以此步骤t++。直到n再也无法被整除为止。此时我们可以思考什么情况下可以操作为1。唯一的可能就是这个数乘N次2后，在除以N次6可以变为1。所以我们可以先除以6，除到不能除后，就除以3，每除一次t要加二（乘一次，除一次）。直到n为1或不能再被3整除为止。如果n操作为了1则输出答案，否则输出-1。

### C) Move Brackets

#include<iostream>

#include<algorithm>

using namespace std;

const int N = 2001;

int main()

{

int t;

cin >> t;

while (t--)

{

int n, ans = 0, num = 0;

char s[N];

cin >> n >> s;

for (int i = 0; i < n; i++)

{

if (s[i] == '(')num++;

else num--;

ans = min(ans, num);

}

cout << -ans << endl;

}

return 0;

}

思路：这道题只要统计没“（”就有“）”的情况。我们只要左括号+右括号-，每次统计最小值，最后输出最小值的相反数就好。

### D) Zero Remainder Array

(以下是Time limit exceeded on test 2的代码)

#include<iostream>

#include<algorithm>

using namespace std;

const int N = 200001;

#define ll long long

struct num

{

ll t, v;

}list[N];

bool cmp(num a, num b)

{

if (a.t > b.t)

return 1;

else if (a.t < b.t)

return 0;

else

{

if (a.v > b.v)

return 1;

else

return 0;

}

}

int main()

{

ios::sync\_with\_stdio(false);

int t;

cin >> t;

while (t--)

{

ll n, k, val[N];

memset(val, 0, sizeof(val));

memset(list, 0, sizeof(list));

cin >> n >> k;

for (int i = 0; i < n; i++)

{

ll temp;

cin >> temp;

val[k - ((temp - 1) % k + 1)]++;

}

for (int i = 0; i < k; i++)

{

list[i].v = i;

list[i].t = val[i];

}

sort(list, list + k, cmp);

int index = list[0].v == 0 ? 1 : 0;

ll ans = list[index].t == 0 ? 0 : (list[index].t - 1) \* k + list[index].v + 1;

cout << ans << endl;

}

return 0;

}

(心情复杂)

(以下是AC代码)

#include<iostream>

#include<algorithm>

#include<map>

using namespace std;

#define ll long long

const int N = 200001;

int main()

{

ios::sync\_with\_stdio(0); cin.tie(0), cout.tie(0);

int t;

cin >> t;

while (t--)

{

ll n, k, ans, index = 0, mx = 0;

map<ll, ll>val;

cin >> n >> k;

for (int i = 0; i < n; i++)

{

ll temp;

cin >> temp;

if (k - temp % k != k) {

val[k - temp % k]++;

if (mx < val[k - temp % k])

{

mx = val[k - temp % k];

index = k - temp % k;

}

else if (mx == val[k - temp % k])index = max(index, k - temp % k);

}

ans = mx != 0 ? (mx - 1) \* k + index + 1 : 0;

}

cout << ans << endl;

}

return 0;

}

思路：这道题真的写到怀疑人生，写了n次都超时，最后看了看题解，原来数组会卡，要用map。那好吧，虽然我对map真的不大熟。说说这道题为了能被k整除，只要把这个数加上k - temp % k就好了（当然也可以在这基础上加任意个k）统计k - temp % k的个数，统计出k - temp % k中次数出现最多的一个（0不算）如果又出现次数相同大的就取（k - temp % k）值大的那个。如果全可被整除ans为0，否则答案为(mx - 1) \* k + index + 1，最后输出答案。

### E1) Reading Books (easy version)

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

#define ll long long

int main()

{

ios::sync\_with\_stdio(0); cin.tie(0), cout.tie(0);

ll n, k;

cin >> n >> k;

vector<ll>a, b, c;

for (ll i = 0; i < n; i++)

{

ll t, x, y; cin >> t >> x >> y;

if (x == 1 && y == 1)

a.push\_back(t);

if (x == 1 && y == 0)

b.push\_back(t);

if (x == 0 && y == 1)

c.push\_back(t);

}

if (b.size() > c.size()) swap(b, c);

if (a.size() + b.size() < k)

cout << -1 << endl;

else

{

sort(b.begin(), b.end());

sort(c.begin(), c.end());

for (ll i = 0; i < b.size(); i++)

{

b[i] += c[i];

a.push\_back(b[i]);

}

sort(a.begin(), a.end());

ll ans = 0;

for (ll i = 0; i < k; i++)

ans += a[i];

cout << ans << endl;;

}

return 0;

}

思路：这道题的大意是两个人都有喜欢的书，且两个人一起读书。每个人都要读够k本喜欢的书，求读完最少的时间。我们可以找到4种书，都喜欢的，都不喜欢的，Alice喜欢的，Bob喜欢的。都不喜欢的可以不要，然后把每个人各喜欢的和都喜欢的分三组存下来。然后从小到大排序，然后把各喜欢的合并并放入都喜欢的里面。再排一次序。最后拿耗时少的书，拿到够就好了，最后输出答案。

### E2) Reading Books (hard version)

这个写不出……

### F) Cyclic Shifts Sorting

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

int ans[252000], a[510], tot;

void change(int index) {

int t1, t2, t3;

t1 = a[index], t2 = a[index + 1], t3 = a[index + 2];

a[index] = t3, a[index + 1] = t1, a[index + 2] = t2;

}

int findMin(int i, int n)

{

int index, mn = 999999;

for (int j = i; j <= n; j++)

if (a[j] < mn)

{

mn = a[j];

index = j;

}

return index;

}

void backToTop(int i, int index)

{

while (index >= i + 2)

{

ans[++tot] = index - 2;

change(index - 2);

index -= 2;

}

if (index == i + 1)

{

change(i);

change(i);

ans[++tot] = i;

ans[++tot] = i;

}

}

int findMax(int i, int n)

{

int index, mx = -1;

for (int j = i; j >= 1; j--)

if (a[j] > mx)

{

mx = a[j];

index = j;

}

return index;

}

void backToBack(int i, int index)

{

while (index <= i - 2)

{

ans[++tot] = index;

change(index);

index += 2;

}

if (index == i - 1)

{

change(i - 2);

ans[++tot] = i - 2;

}

}

void detect(int n)

{

for (int i = 1; i < n; i++)

if (a[i] > a[i + 1])

tot = -1;

}

int main() {

int t, n, index;

cin >> t;

while (t--) {

tot = 0;

cin >> n;

for (int i = 1; i <= n; i++)

cin >> a[i];

for (int i = 1; i <= n - 2; i++)

{

index = findMin(i, n);

backToTop(i, index);

}

for (int i = n; i >= 3; i--)

{

index = findMax(i, n);

backToBack(i, index);

}

detect(n);

if (tot == -1)

cout << -1 << endl;

else

{

cout << tot << endl;

for (int i = 1; i <= tot; i++)

cout << ans[i] << " ";

cout << endl;

}

}

return 0;

}

思路：这道题就是提供一种操作，然后要你把一个数组从小到大排序。一开始我的思路是类似于冒泡排序，不过做法是每次找最小的，然后把数操作到数组的最前面，操作到最后剩3个数，对这三个数进行操作，如果操作不出来则输出-1，可以就输出答案，结果是错的。看看题解，发现还要从后到前排回去，把大数后排。最后排回前面是在判断能否操作成从小到大。如果不行则输出-1，可以就输出答案。

### 赛后总结：

这次比较好，事后补题就一题补不出来，hard version思路有点乱还没调出一个案例，其他最后还是做出来了。D题真的是写到我怀疑人生，调了一个多小时才知道会卡数组（无比难受），结果要用map然后map我又不大熟（怀疑人生）。

## 3) Codeforces Round #640 (Div. 4)

### A) Sum of Round Numbers

#include<iostream>

using namespace std;

const int N = 20000;

int main()

{

std::ios::sync\_with\_stdio(false); std::cin.tie(0); std::cout.tie(0);

int t;

cin >> t;

while (t--)

{

int n, sum = 0, num[N], i = 0, b = 1;

cin >> n;

while (n / 10 != 0)

{

if (n % 10 == 0)

{

n /= 10;

b \*= 10;

}

else

{

sum++;

num[i++] = n % 10 \* b;

n /= 10;

b \*= 10;

}

}

num[i] = n \* b;

sum++;

cout << sum << endl;

for (int j = 0; j < sum; j++)

cout << num[j] << " ";

cout << endl;

}

return 0;

}

思路：这道题大致就是要我们把某个数字每一位拆开，并统计有多少个数（如果是0就不用了）每次求余得到一位数，如果余数零就跳过，不是0则把该数乘上相应的位数，sum++最后输出结果。

### B) Same Parity Summands

#include<iostream>

using namespace std;

#define ll long long

const int N = 20000;

int main()

{

std::ios::sync\_with\_stdio(false); std::cin.tie(0); std::cout.tie(0);

int t;

cin >> t;

while (t--)

{

ll n, k;

cin >> n >> k;

if (n < k)

cout << "NO" << endl;

else

{

if (n < 2 \* k)

{

if ((n - k + 1) % 2 == 0)

cout << "NO" << endl;

else

{

cout << "YES" << endl;

for (int i = 0; i < k - 1; i++)

cout << 1 << " ";

cout << n - k + 1 << endl;

}

}

else

{

if ((n - k + 1) % 2 == 0)

{

if ((n - (k - 1) \* 2) % 2 == 0)

{

cout << "YES" << endl;

for (int i = 0; i < k - 1; i++)

cout << 2 << " ";

cout << n - (k - 1) \* 2 << endl;

}

else

cout << "NO" << endl;

}

else

{

cout << "YES" << endl;

for (int i = 0; i < k - 1; i++)

cout << 1 << " ";

cout << n - k + 1 << endl;

}

}

}

}

return 0;

}

思路：这道题我的思路感觉还是比较奇葩的。因为我们要所有元素都为偶数或都为奇数所以，我把k-1个元素全变为1或2，看对应是否元素会变为奇数和偶数，如果又符合条件的就输出，如果没有符合条件的就输出-1。

### C) K-th Not Divisible by n

#include<iostream>

using namespace std;

#define ll long long

const int N = 20000;

int main()

{

std::ios::sync\_with\_stdio(false); std::cin.tie(0); std::cout.tie(0);

int t;

cin >> t;

while (t--)

{

ll n, k; cin >> n >> k;

if (k % (n - 1) == 0)

cout << k / (n - 1) \* n - 1 << endl;

else

cout << k / (n - 1) \* n + k % (n - 1) << endl;

}

return 0;

}

思路：这道题要找一下规律，找到规律以后就很好做了。

### D) Alice, Bob and Candies

#include<iostream>

using namespace std;

#define ll long long

const int N = 20001;

int main()

{

std::ios::sync\_with\_stdio(false); std::cin.tie(0); std::cout.tie(0);

int t;

cin >> t;

while (t--)

{

int step = 0, cura = 0, curb = 0, i = 1, j, suma = 0, sumb = 0, n, a[N], flag = 1, used[N];

cin >> n;

memset(a, 0, sizeof(a));

memset(used, 0, sizeof(used));

for (int k = 1; k <= n; k++)

cin >> a[k];

j = n;

if (1 == n)

cout << 1 << " " << a[1] << " " << 0 << endl;

else

{

while (i <= j)

{

if (flag == 1)

{

while (cura <= curb)

{

if (used[i] == 1)

{

flag = 2;

break;

}

cura += a[i];

suma += a[i];

used[i] = 1;

i++;

}

curb = 0;

flag = 0;

step++;

}

else if (flag == 0)

{

while (curb <= cura)

{

if (used[j] == 1)

{

flag = 2;

break;

}

curb += a[j];

sumb += a[j];

used[j] = 1;

j--;

}

cura = 0;

flag = 1;

step++;

}

else

break;

}

cout << step << " " << suma << " " << sumb << endl;

}

}

return 0;

}

思路：这道题我写得有点乱，大致思路是这样子的，AB两人，一个从左到右吃糖，一个从右往左吃糖，每次记录当前吃糖的重量，一方吃糖数大于另一方则换一方吃糖，一直下去，直到两边相遇，就结束。

### E) Special Elements

#include<iostream>

using namespace std;

const int N = 8001;

int main()

{

std::ios::sync\_with\_stdio(false); std::cin.tie(0); std::cout.tie(0);

int t;

cin >> t;

while (t--)

{

int x, n, res = 0, sum[N], cnt[N];;

memset(cnt, 0, sizeof(cnt));

memset(sum, 0, sizeof(sum));

cin >> n;

for (int i = 1; i <= n; i++)

{

cin >> x;

sum[i] = sum[i - 1] + x;

cnt[x] ++;

}

for (int i = 1; i < n; i++)

{

for (int j = i + 1; j <= n; j++)

{

int tmp = sum[j] - sum[i - 1];

if (tmp < N) {

res += cnt[tmp];

cnt[tmp] = 0;

}

}

}

cout << res << endl;

}

return 0;

}

思路：这道题用前缀和做，先记录每个元素的个数，然后枚举起点和终点，求得两点间的和，然后找有没有匹配的数字，有则把这个数的个数加入答案中，同时要记得把这个数字的个数清零，免得重复加，最后输出答案。

### F) Binary String Reconstruction

#include<iostream>

using namespace std;

const int N = 8001;

int main()

{

std::ios::sync\_with\_stdio(false); std::cin.tie(0); std::cout.tie(0);

int t;

cin >> t;

while (t--)

{

int n0, n1, n2, flag = 0;

cin >> n0 >> n1 >> n2;

if (0 == n1)

{

if (0 == n2)

for (int i = 0; i <= n0; i++) cout << '0';

else

for (int i = 0; i <= n2; i++) cout << '1';

}

else

{

for (int i = 0; i <= n0; i++) cout << '0';

for (int i = 0; i <= n2; i++) cout << '1';

n1--;

while (n1 != 0)

{

char temp = flag == 0 ? '0' : '1';

cout << temp;

flag = flag == 0 ? 1 : 0;

n1--;

}

}

cout << endl;

}

return 0;

}

思路：这道题我的思路是首先看n1是否为0，是的话要么n0为0，或n2为0（同时不为0的话n1不可能0）。只有n2或n0的话，按要求输出就好了。n1不为0的话，先把“00”处理完，然后把“11”处理完，此时就剩“01”或“10”的情况。因为“11”与“00”的衔接就有“01”了所以“01”--。接下来看看还剩多少个，然后0101的加下去，知道够数为止。

### G) Special Permutation

#include<iostream>

using namespace std;

int main()

{

std::ios::sync\_with\_stdio(false); std::cin.tie(0); std::cout.tie(0);

int t;

cin >> t;

while (t--)

{

int n;

cin >> n;

if (n < 4)

cout << -1 << endl;

else

{

int num = n;

while (num != 3 && num != 4)

{

cout << num << " ";

num -= 2;

}

cout << "3 1 4 2 ";

num = num == 3 ? 6 : 5;

while (num < n)

{

cout << num << " ";

num += 2;

}

cout << endl;

}

}

return 0;

}

思路：n小于4时没有解决办法，等于4时，3 1 4 2。其他情况的话，把奇数和偶数分别放两边就好了，两两之间差2，符合条件，最后输出。

### 赛后总结：

感天动地，第一次能把题目全部补齐（心情激动），但想想这只是div 4，瞬间感觉人又不好了（我太菜了555）。div 4难度感觉还好（因为它是d4适合我这种蒟蒻），还是能写的出来的，就是有一些题一时间没有思路，或者不够好，比如E题一开始没想到前缀和，乱七八糟搞了半天没搞出开。

## 4）Codeforces Round #650 (Div. 3)

### Short Substrings

#include<iostream>

#include<cstring>

#include<algorithm>

using namespace std;

const int N = 101;

char a[N], b[N];

int main()

{

int t;

cin >> t;

while (t--)

{

int i;

cin >> b;

a[0] = b[0];

for (i = 1; 2 \* i - 1 < strlen(b); i++)

a[i] = b[2 \* i - 1];

for (int j = 0; j < i; j++)

cout << a[j];

cout << endl;

}

return 0;

}

思路：这道题目花里胡哨，思路其实很简单，就是把第一个字符（下标为0）存下来，然后把字符串的奇数位存进来，最后输出存下来的字符。

### Even Array

#include<iostream>

#include<cstring>

#include<algorithm>

using namespace std;

const int N = 41;

int num[N];

int main()

{

int t;

cin >> t;

while (t--)

{

int n, e = 0, o = 0, ans = 0;

cin >> n;

for (int i = 0; i < n; i++)

{

cin >> num[i];

if (num[i] % 2 != i % 2)

{

if (num[i] % 2 == 0)

e++;

else

o++;

}

ans = e == o ? e : -1;

}

cout << ans << endl;

}

return 0;

}

思路：这道题其实就是看看下标与数组值是否同奇偶，不是，若值为奇数则o++，为偶数就e++，最后看e和o是否相等，因为只有错位奇数与错位偶数的数量相同才能进行交换，相等就输出e或o的值（都一样），若不相等则输出-1表示没有交换方案。

### Social Distance

#include<iostream>

#include<cstring>

#include<algorithm>

using namespace std;

const int N = 200001;

char str[N];

int main()

{

int t;

cin >> t;

while (t--)

{

int n, k;

cin >> n >> k;

cin >> str;

if (n==0)

{

cout << 0 << endl;

continue;

}

int sum = 0, c1 = 0, c0 = 0;

for (int i = 0; i < n; i++)

{

if (str[i] == '1')

{

c1++;

if (c1 == 1)

sum += c0 / (k + 1);

else

{

sum += (c0 - k) / (k + 1);

c1 = 1;

}

c0 = 0;

}

else

c0++;

}

if (str[n - 1] == '0' && c1 != 0)

sum += c0 / (k + 1);

else

sum += (c0 + k) / (k + 1);

cout << sum << endl;

}

return 0;

}

思路：这道题主要意思是，尽量的把0转成1，使得在1与1间隔在符合的条件下尽可能的让1的数量增多，并输出转换的个数。 思路就是把所给的字符遍历一遍，遇到0则c0++遇到1的话，c0清零，同时看看这一段0中最多能转换多少个0为1，同时c1++。最后还要处理一下，全程没有1的字符段要特殊处理，最后输出结果。

### 赛后总结：

这场比赛是今天打的（8月21号）所以一还没来得及补，晚点再一起补，开场三道还好C题花了点时间来理解，至于D题我到现在都还没有想出思路，晚点再补。

# 最终总结：

这次考核算是挑战了一些以前不敢动的算法。以前看动态规划，kmp等题真的碰都不敢碰，一开始看它的模板代码时一脸懵逼。这次考核的知识点挺多的，因为时间原因，很多算法还没有时间来学，日后慢慢补，而且这次我选的题比较偏基础（难的驾驭不了[哭笑]）。以后再继续好好刷题。比较可惜的是时间有限（还有前端考核要做）这个假期的比赛打的比较少，后面补体题只有div4补完了，div3的hard version普遍补不出来。至于刷CF过程可谓艰苦(我太菜了)。上来看题看不懂，看懂了题没思路，有思路后又不知怎么写代码，代码写好了一堆bug，bug改好了从test1wa到testn，wa改好了以后又变成rle，实在没办法去看题解，结果题解一堆看不懂，看懂了以后写好了代码，明明与题解相似，结果答案大不一样（人都傻）。日后好好刷题，不然太尴尬了。总体来说，这次考核收获还是很多的，学到了不少算法，特别是stl以前基本没用过，现在突然发现里面的东西真香。