### KMP-search

void prefix\_table(char pattern[], int prefix[], int n) {  
 prefix[0] = 0;  
 int len = 0;  
 int 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] = 0;  
 i++;  
 }  
 }  
 }  
}  
void move\_prefix\_table(int prefix[], int n) {  
 for (int i = n-1; i > 0; i--)  
 prefix[i] = prefix[i-1];  
 prefix[0] = -1;  
}  
void kmp\_search(char text[], char pattern[]) {  
 int n = strlen(pattern);  
 int m = strlen(text);  
 int\* prefix = (int \*)malloc(sizeof(int) \* n);  
 prefix\_table(pattern, prefix, n);  
 move\_prefix\_table(prefix, n);  
 int j = 0;  
 int i = 0;  
 while (i < m) {  
 if (j == n-1 && text[i] == pattern[j]) {  
 printf("Found pattern at %d\n", i - j);  
 j = prefix[j];  
 }  
 if (text[i] == pattern[j])  
 i++, j++;  
 else {  
 j = prefix[j];  
 if (j == -1)  
 i++, j++;  
 }  
 }  
 free(prefix);  
}

void prefix\_table(string pattern, int prefix[]) {  
 prefix[0] = 0;  
 int len = 0;  
 int i = 1;  
 while (i < (int)pattern.length()) {  
 if (pattern[i] == pattern[len]) {  
 ++len;  
 prefix[i] = len;  
 ++i;  
 }  
 else {  
 if (len > 0) len = prefix[len - 1];  
 else prefix[i++] = 0;  
 }  
 }  
}  
void move\_prefix\_table(int prefix[], int n) {  
 for (int i = n - 1; i > 0; --i)  
 prefix[i] = prefix[i-1];  
 prefix[0] = -1;  
}  
void kmp\_search(string text, string pattern) {  
 int n = pattern.length();  
 int m = text.length();  
 int\* prefix = (int \*)malloc(sizeof(int) \* n);  
 prefix\_table(pattern, prefix);  
 move\_prefix\_table(prefix, n);  
 // pattern[j] , len(pattern) = n  
 // text[i] , len(text) = m  
 int j = 0;  
 int i = 0;  
 while (i < m) {  
 if (j == n-1 && text[i] == pattern[j]) {  
 printf("Found pattern at %d\n", i - j);  
 j = prefix[j];  
 }  
 if (text[i] == pattern[j])  
 ++i, ++j;  
 else {  
 j = prefix[j];  
 if (j == -1)  
 ++i, ++j;  
 }  
 }  
 free(prefix);  
}

### Manacher

vector<int> radius, let;  
string expa\_str;  
void Manacher(const string &str, int &pos, int &max\_len) {  
 int orig\_len = str.length();  
 int expa\_len = (orig\_len + 1) << 1;  
 max\_len = 0;  
 radius.resize(expa\_len + 1);  
 expa\_str.resize(expa\_len + 1);  
 //@#0#1#2#3#4#5#6#7#8#9#$  
 expa\_str[0] = '@';  
 expa\_str[1] = '#';  
 expa\_str[expa\_len] = '$';  
 for (int i = 1; i <= orig\_len; ++i) {  
 expa\_str[i << 1] = str[i-1];  
 expa\_str[i << 1 | 1] = '#';  
 }  
 radius[1] = 1;  
 for (int max\_R = 0, center = 0, i = 2; i < expa\_len; ++i) {  
 radius[i] = i < max\_R ? min(max\_R-i, radius[2\*center-i]) : 1;  
 while (expa\_str[i-radius[i]] == expa\_str[i+radius[i]])  
 ++radius[i];  
 if (radius[i] + i > max\_R) {  
 max\_R = radius[i] + i;  
 center = i;  
 }  
 if (radius[i]-1 > max\_len) {  
 max\_len = radius[i]-1;  
 pos = (center - radius[i] + 1) << 1;  
 }  
 }  
}  
  
//odd为false，字符串为奇回文串  
int start\_mid(int x, bool odd) {  
 return odd ? radius[(x+1) << 1] - 1 : radius[(x+1) << 1 | 1] - 1;  
}  
  
//知道回文左边界，且在Manacher函数运行结束后使用  
int start\_left(int x, string str) {  
 int expand\_len = (str.length() + 1) << 1;  
 let.resize(expand\_len + 1);  
 for (int i = 0; i <= expand\_len; i++)  
 let[i] = 0;  
 for (int i = 2; i < expand\_len; i++)  
 if (let[i - radius[i] + 1] < i + 1)  
 let[i - radius[i] + 1] = i + 1;  
 for (int i = 1; i <= expand\_len; i++)  
 if (let[i] < let[i - 1])  
 let[i] = let[i - 1];  
 return let[(x + 1) << 1] - ((x + 1) << 1);  
}

### 欧几里德算法

template<typename T>  
T Gcd(T num1, T num2) {  
 return !num2 ? num1 : Gcd(num2, num1 % num2);  
}

### 欧拉筛法

#include <cstdio>  
const int maxn = 5e8;  
int seek = 0;  
int prime[maxn];  
bool sifter[maxn];  
//bool\* sifter = (bool \*)memset(sifter, 0, sizeof(sifter));  
  
void sievePrime() {  
 for (int i = 2; i <= maxn; i++) {  
 if (sifter[i] == false)  
 prime[seek++] = i;  
  
 for (int j = 0; j < seek; j++) {  
 if (i \* prime[j] > maxn) break;  
  
 sifter[i \* prime[j]] = true;  
  
 if (!(i % prime[j])) break;  
 }  
 }  
}  
  
int main()  
{  
 sievePrime();  
 for (int i = 0; i < seek; i++)  
 printf("%d ", prime[i]);  
 return 0;  
}

### 乘法逆元

long long binPow(long long base, long long expo) {  
 long long res = 1;  
 while (expo != 0) {  
 if (expo & 1)  
 res = (1LL \* res \* base) % mod;  
  
 base = (1LL \* base \* base) % mod;  
 expo >>= 1;  
 }  
 return res;  
}  
  
long long inv(long long x) {  
 return binPow(x, mod-2, mod);  
}  
  
void Inv() {  
 inv[1] = 1;  
 for (int i = 2; i < maxn; i++)  
 inv[i] = (mod - mod/i) % mod \* inv[mod%i] % mod;  
}

### 组合数

int C(int n, int k, const vector <long long> &fact, const vector <long long> &inv) {  
 if (k > n) return 0;  
 int multiply = (1LL \* fact[n] \* inv[k]) % MOD;  
 multiply = (1LL \* multiply \* inv[n - k]) % MOD;  
 return multiply;  
}  
  
vector <long long> fact(n + 1, 1LL);  
vector <long long> inv(n + 1, 1LL);  
for (int i = 1; i <= n; ++i) {  
 fact[i] = (fact[i - 1] \* i) % MOD;  
 inv[i] = binPow(fact[i], MOD - 2);  
}  
  
int countLess = C(can\_less, cnt\_less, fact, inv);  
int countBig = C(can\_big, cnt\_big, fact, inv);

#include <bits/stdc++.h>  
using namespace std;  
const int MOD = 1e9 + 7;  
  
int binPow(int a, int n) {  
 int res = 1;  
 while (n) {  
 if (n & 1)  
 res = (1LL \* res \* a) % MOD;  
 a = (1LL \* a \* a) % MOD;  
  
 n >>= 1;  
 }  
 return res;  
}  
  
void binarySearch(int n, int x\_position, int &cnt\_big, int &cnt\_less) {  
 int left = 0, right = n;  
  
 while(left < right) {  
 int middle = (left + right) / 2;  
 if (x\_position >= middle) {  
 if (x\_position != middle) cnt\_less++;  
 left = middle + 1;  
 }  
 else if (x\_position < middle){  
 cnt\_big++;  
 right = middle;  
 }  
 }  
}  
  
int C(int n, int k, const vector <long long> &fact, const vector <long long> &inv) {  
 if (k > n) return 0;  
 int multiply = (1LL \* fact[n] \* inv[k]) % MOD;  
 multiply = (1LL \* multiply \* inv[n - k]) % MOD;  
 return multiply;  
}  
  
int main() {  
 ios\_base::sync\_with\_stdio(false);  
 cin.tie(nullptr);  
  
 int n, x, x\_position;  
 long long ans = 0;  
  
  
 cin >> n >> x >> x\_position;  
 vector <long long> fact(n + 1, 1LL);  
 vector <long long> inv(n + 1, 1LL);  
 for (int i = 1; i <= n; ++i) {  
 fact[i] = (fact[i - 1] \* i) % MOD;  
 inv[i] = binPow(fact[i], MOD - 2);  
 }  
  
 int cnt\_big = 0, cnt\_less = 0;  
 binarySearch(n, x\_position, cnt\_big, cnt\_less);  
  
 int other = (n - cnt\_big - cnt\_less - 1);  
 int can\_big = n - x, can\_less = x - 1;  
  
 int countLess = C(can\_less, cnt\_less, fact, inv);  
 int countBig = C(can\_big, cnt\_big, fact, inv);  
  
 countBig = (1LL \* countBig \* fact[cnt\_big]) % MOD;  
 countLess = (1LL \* countLess \* fact[cnt\_less]) % MOD;  
  
 int multiply = (1LL \* countBig \* countLess) % MOD;  
 multiply = (1LL \* multiply \* fact[other]) % MOD;  
  
 ans = (ans + multiply) % MOD;  
 cout << ans << endl;  
 return 0;  
}

### 木棍dfs剪枝

#include <cstdio>  
#include <algorithm>  
#include <cstring>  
using namespace std;  
const int maxn = 105;  
int sticks[maxn];  
int vis[maxn]; /\*记录每根木棍的访问\*/  
int n; /\*多组输入\*/  
int max\_len; /\*假设最长木棍长度\*/  
int max\_cnt; /\*最大木棍数量\*/  
  
/\*  
 \* 正在拼接第stick根原始木棍(已经拼好了stick-1根)  
 \* 第stick根木棍的当前长度为now\_len  
 \* 拼接到第stick根木棍中的上一根小木棍为last  
 \*/  
bool DFS(int stick, int now\_len, int last) {  
 /\*所有原始木棍都已拼好，搜索成功\*/  
 if (stick > max\_cnt)  
 return true;  
 /\*第stick根木棍已经拼好，去拼下一根\*/  
 if (now\_len == max\_len)  
 return DFS(stick+1, 0, 1);  
  
 /\*记录尝试向当前原始木棍拼接的最近的失败的木棍长度\*/  
 int record = 0;  
 for (int i = last; i <= n; ++i)  
 if (!vis[i] && now\_len + sticks[i] <= max\_len && record != sticks[i]) {  
 vis[i] = 1;  
 if (DFS(stick, now\_len + sticks[i], i + 1))  
 return true;  
 record = sticks[i];  
 vis[i] = 0; /\*还原现场\*/  
 /\*贪心，再用1根木棍恰好拼完当前原始木棍必然比再用若干根木棍拼完更好\*/  
 if (now\_len == 0 || now\_len + sticks[i] == max\_len)  
 return false;  
 }  
 /\*所有分支均尝试过，搜索失败\*/  
 return false;  
}  
  
int main()  
{  
 while (scanf("%d", &n) && n) {  
 int sum = 0;  
 int val = 0;  
 for (int i = 1; i <= n; ++i) {  
 scanf("%d", &sticks[i]);  
 sum += sticks[i];  
 val = max(sticks[i], val);  
 }  
 sort(sticks+1, sticks+1+n);  
 reverse(sticks+1, sticks+1+n);  
 for (int max\_len = val; max\_len <= sum; ++max\_len) {  
 if (sum % max\_len)  
 continue;  
 max\_cnt = sum / max\_len;  
 memset(vis, 0, sizeof(vis));  
 if (DFS(1, 0, 1))  
 break;  
 }  
 printf("%d\n", max\_len);  
 }  
 return 0;  
}

### segment-tree

#include<stdio.h>  
#define MAX\_LEN 1000  
  
void bulid\_tree(int arr[], int tree[], int node, int start, int end) {  
 if (start == end) {  
 tree[node] = arr[start];  
 }  
 else {  
 int mid = (start + end) / 2;  
 int left\_node = 2 \* node + 1;  
 int right\_node = 2 \* node + 2;  
  
 bulid\_tree(arr, tree, left\_node, start, mid);  
 bulid\_tree(arr, tree, right\_node, mid+1, end);  
 tree[node] = tree[left\_node] + tree[right\_node];  
 }  
}  
  
void update\_tree(int arr[], int tree[], int node, int start, int end, int idx, int val) {  
 if (start == end) {  
 arr[idx] = val;  
 tree[node] = val;  
 }  
 else {  
 int mid = (start + end) / 2;  
 int left\_node = 2 \* node + 1;  
 int right\_node = 2 \* node + 2;  
 if (idx >= start && idx <= mid) {  
 update\_tree(arr, tree, left\_node, start, mid, idx, val);  
 }  
 else {  
 update\_tree(arr, tree, right\_node, mid+1, end, idx, val);  
 }  
 tree[node] = tree[left\_node] + tree[right\_node];  
 }  
}  
  
int query\_tree(int arr[], int tree[], int node, int start, int end, int L, int R) {  
 printf("start = %d\n", start);  
 printf("end = %d\n", end);  
 printf("\n");  
  
 if (R < start || L > end) {  
 return 0;  
 }  
 else if (L <= start && end <= R) {  
 return tree[node];  
 }  
 else if (start == end) {  
 return tree[node];  
 }  
 else {  
 int mid = (start + end) / 2;  
 int left\_node = 2 \* node + 1;  
 int right\_node = 2 \* node + 2;  
 int sum\_left = query\_tree(arr, tree, left\_node, start, mid, L, R);  
 int sum\_right = query\_tree(arr, tree, right\_node, mid+1, end, L, R);  
 return sum\_left + sum\_right;  
 }  
}  
  
int main()  
{  
 int arr[] = {1, 3, 5, 7, 9, 11};  
 int size = 6;  
 int tree[MAX\_LEN] = {0};  
  
 bulid\_tree(arr, tree, 0, 0, size-1);  
  
 int i;  
 for(i = 0; i <= 14; i++) {  
 printf("tree[%d] = %d\n", i, tree[i]);  
 }  
  
 printf("\n");  
 update\_tree(arr, tree, 0, 0, size-1, 4, 6);  
 for(i = 0; i <= 14; i++) {  
 printf("tree[%d] = %d\n", i, tree[i]);  
 }  
  
 printf("\n");  
 int s = query\_tree(arr, tree, 0, 0, size-1, 2, 5);  
 printf("s = %d\n", s);  
  
 return 0;  
}

### 并查集

#include<stdio.h>  
#include<stdlib.h>  
#define VERTICES 6  
  
void initialize(int parent[], int rank[]) {  
 int i;  
 for (i = 0; i < VERTICES; i++) {  
 parent[i] = -1;  
 rank[i] = 0;  
 }  
}  
  
int find\_root(int x, int parent[]) {  
 int x\_root = x;  
 while (parent[x\_root] != -1) {  
 x\_root = parent[x\_root];  
 }  
 return x\_root;  
}  
  
/\* 1 - union successfully, 0 - union failed \*/  
int union\_vertices(int x, int y, int parent[], int rank[]) {  
 int x\_root = find\_root(x, parent);  
 int y\_root = find\_root(y, parent);  
  
 if (x\_root == y\_root) {  
 return 0;  
 }  
 else {  
 //parent[x\_root] = y\_root;  
 if (rank[x\_root] > rank[y\_root]) {  
 parent[y\_root] = x\_root;  
 }  
 else if (rank[y\_root] < rank[x\_root]) {  
 parent[x\_root] = y\_root;  
 }  
 else {  
 parent[x\_root] = y\_root;  
 rank[y\_root]++;  
 }  
 return 1;  
 }  
}  
  
int main()  
{  
 int parent[VERTICES] = {0};  
 int rank[VERTICES] = {0};  
 int edges[6][2] = {  
 {0, 1}, {1, 2}, {1, 3},  
 {2, 4}, {3, 4}, {2, 5}  
 };  
  
 initialize(parent, rank);  
 int i;  
 for (i = 0; i < 6; i++) {  
 int x = edges[i][0];  
 int y = edges[i][1];  
 if (union\_vertices(x, y, parent, rank) == 0) {  
 printf("Cycle detected!\n");  
 exit(0);  
 }  
 }  
 printf("No cycles fond.\n");  
 return 0;  
}

### 进制转换

#include <cstdio>  
#include <vector>  
using namespace std;  
  
// B：bit 二进制  
// T：ternary 三进制  
// Q：quaternary 四进制  
// O：octonary 八进制  
// D：decimal 十进制  
// H：hexadecimal 十六进制  
  
// 将一个P进制的数转换为D进制的数  
int anytoD(int num\_P, int base\_P) {  
 int num\_D = 0;  
 int weight = 1;  
 while (num\_P != 0) {  
 num\_D += (num\_P % 10) \* weight;  
 num\_P /= 10;  
 weight \*= base\_P;  
 }  
 return num\_D;  
}  
  
// 将一个D进制的数转换为C进制的数  
vector<int> numto;  
void Dtoany(int num\_D, int base\_C) {  
 do {  
 numto.push\_back(num\_D % base\_C);  
 num\_D /= base\_C;  
 }while (num\_D != 0);  
}  
  
int main()  
{  
 int num\_P = 101110011;  
 int base\_P = 2;  
 int base\_C = 8;  
 int num\_D = anytoD(num\_P, base\_P);  
 Dtoany(num\_D, base\_C);  
 for (auto it = numto.end()-1; it != numto.begin()-1; --it)  
 printf("%d", \*it);  
 return 0;  
}

### 归并排序

#include <cstdio>  
#include <cstring>  
  
template<typename T>  
void mergeSort(T arr[], T reg[], int start, int end) {  
 if (start >= end)  
 return;  
 int len = end - start, mid = (len >> 1) + start;  
 int start1 = start, end1 = mid;  
 int start2 = mid + 1, end2 = end;  
 mergeSort(arr, reg, start1, end1);  
 mergeSort(arr, reg, start2, end2);  
 int k = start;  
 while (start1 <= end1 && start2 <= end2)  
 reg[k++] = arr[start1] <= arr[start2] ? arr[start1++] : arr[start2++];  
 while (start1 <= end1)  
 reg[k++] = arr[start1++];  
 while (start2 <= end2)  
 reg[k++] = arr[start2++];  
 for (k = start; k <= end; k++)  
 arr[k] = reg[k];  
}  
  
long long res = 0;  
template<typename T>  
void reverse\_arr(T arr[], T reg[], int start, int end) {  
 if (start >= end)  
 return;  
 int len = end - start, mid = (len >> 1) + start;  
 int start1 = start, end1 = mid;  
 int start2 = mid + 1, end2 = end;  
 reverse\_arr(arr, reg, start1, end1);  
 reverse\_arr(arr, reg, start2, end2);  
 int k = start;  
 while (start1 <= end1 && start2 <= end2) {  
 if (arr[start1] <= arr[start2])  
 reg[k++] = arr[start1++];  
 else {  
 reg[k++] = arr[start2++];  
 res += (mid - start1 + 1);  
 }  
 }  
 while (start1 <= end1)  
 reg[k++] = arr[start1++];  
 while (start2 <= end2)  
 reg[k++] = arr[start2++];  
 for (k = start; k <= end; k++)  
 arr[k] = reg[k];  
}  
  
int main()  
{  
 int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};  
 int tem[10];  
 memset(tem, 0, sizeof(tem));  
 reverse\_arr(arr, tem, 0, 9);  
 return 0;  
}

### 快速幂

#include <cstdio>  
const int mod = 99991;  
  
template<typename T>  
T binPow(T base, T expo) {  
 T res = 1;  
 while (expo != 0) {  
 if (expo & 1)  
 res = (1LL \* res \* base) % mod;  
  
 base = (1LL \* base \* base) % mod;  
 expo >>= 1;  
 }  
 return res;  
}  
  
int main()  
{  
 long long base = 23, expo = 19898283988388888;  
 long long res;  
 for (int i = 0; i < 1000000; ++i)  
 res = binPow(base, expo);  
  
 printf("%lld\n", res);  
 return 0;  
}

### 自定义模板

#include <bits/stdc++.h>  
using namespace std;  
typedef long long ill;  
typedef usigned long long ull;  
#define bug printf("<------>\n");  
  
const int inf = 0x3f3f3f3f;  
//memset(arr, inf, sizeof(arr));  
const double pi = acos(-1.0);  
const double eps = 1e-6;  
const int dir[][2]={{0, 1}, {1, 0}, {0, -1}, {-1, 0}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};  
  
Node() {}  
Node(int \_node1, double \_node2, char \_node3): node1(\_node1), node2(\_node2), node3(\_node3) {}  
<++>  
  
int main() {  
#ifndef ONLINE\_JUDGE  
 freopen("in.txt", "r", stdin);  
 freopen("out.txt", "w", stdout);  
#endif  
 //ios::sync\_with\_stdio(false);  
 //cin.tie(NULL), cout.tie(NULL);  
 solve();  
 return 0;  
}

### 格式化读取输出

//sscanf 函数将多个不同类型的变量写入到字符串类型的一个变量中  
//sprintf 函数将一个字符串类型的变量写入到多个不同类型的变量中  
//两者不具备输入与输出功能，而是具有写入到其他变量中的功能  
#include<iostream>  
using namespace std;  
int main()  
{  
 int n;  
 double db;  
 char str1[20] = "12345";  
 char str2[20];  
 char str3[20] = "2048:3.14,hello";  
 char str4[20] = "good";  
//自左至右  
 sscanf(str1, "%d", &n); //此时 str1 中的字符串转换为整型  
 printf("%d\n", n); //输出12345  
//自右向左  
 sprintf(str2, "%d", n); //把之前 n 中的12345以字符串的方式放在str2中  
 printf("%s\n", str2); //输出字符串12345  
//复杂格式输入输出  
 sscanf(str3, "%d:%lf,%s", &n, &db, str2);  
 printf("n = %d, db = %.2f, str2 = %s\n", n, db, str2);

sprintf(str2, "%d:%.2f,%s", n, db, str4);  
 printf("str2 = %s\n", str2);  
 return 0;  
}

### STL

vector

* push\_back
* pop\_back
* size
* clear
* insert vec.insert(it, x);(O(N))
* erase vec.erase(vec.begin() + 3);(O(N)) vec.erase(vec.begin(), vec.end());(O(N))
* reserve
* resize

string

* size
* length
* insert str.insert(pos, str1); str.insert(it, it*begin, it*end); str1.insert(str1.begin()+3, str2.begin()+1, str2.end())
* erase str.erase(it); str.erase(first, last); str.erase(pos, len);
* clear str.clear();(O(1))
* substr str.substr(pos, len);(O(len))
* string::npos 用以作为find函数失配时的返回值
* find str.find(str2);(O(nm)) str.find(str2, str.find(str2) + 1);(O(nm))
* replace str.replace(pos, len, str2);(O(str.length())) str.replace(it*begin, it*end, str2);

set  
unordered\_set

* insert st.insert(x);(O(logN))
* find st.find(x);(O(logN)) 没找到返回st.end()
* erase st.erase(it);(O(1)) st.erase(x);(O(logN)) st.erase(first, last);(O(last-first))
* size
* clear (O(N))

map  
unordered\_map

* find mp.find(key);(O(logN)) 没找到返回mp.end()
* erase mp.erase(it);(O(1)) mp.erase(key);(O(logN)) mp.erase(first, last);(O(last-first))
* size
* clear (O(N))

queue

* push
* front
* back
* pop
* empty
* size

stack

* push
* top
* pop
* empty
* size

pair

algorithm

* max
* min
* abs 只能对整数取绝对值
* swap
* reverse reverse(it, it+1)
* next*permutation next*permutation(arr, arr+n); next\_permutation(it, it+n);
* fill fill(arr, arr+n, 0);
* sort
* lower*bound [begin, end)范围内第一个小于等于val的元素的位置 lower*bound(arr, arr+n, val);返回指针 lower\_bound(it, it+n, val);返回迭代器 没有找到返回可以插入的位置(O(log(begin-end)))
* upper*bound [begin, end)范围内第一个大于val的元素的位置 lower*bound(arr, arr+n, val);返回指针 lower\_bound(it, it+n, val);返回迭代器 没有找到返回可以插入的位置(O(log(begin-end)))