

ACM 模板

ACM Template

rogeryoung

目录

1	上号	2
1.1	头文件	2
1.2	预编译	2
1.3	进制转换	2
1.4	常见技巧	3
1.5	快速幂	3
1.6	矩阵快速幂	3
1.7	快速排序	4
1.8	第 k 大数	4
1.9	链表	4
2	数论	6
2.1	GCD 和 LCM	6
2.2	EXGCD	6
2.3	Eratosthenes 筛	6
2.4	Eular 筛	6
2.5	素性测试	7
3	图论	8
3.1	链式前项星	8
3.2	Dijkstra	8
3.3	Bellman-Ford	9
3.4	Floyd	9
3.5	最近公共祖先 (LCA)	9
4	动态规划	11
4.1	背包	11
4.2	最长公共上升序列	12
4.3	数字计数	12

第 1 章 上号

1.1 头文件

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define _fora(i,a,n) for(ll i=(a);i<=(n);i++)
5 #define _forz(i,a,n) for(ll i=(a);i>=(n);i--)
6 #define _forb(i,a) for(ll i=(a);i>0;i-=i&(-i))
7 #define _dbg(x) cout<<"[Log] "<<#x<<" = "<<x<<endl;
8 #define _in(i,min,max) ( ((i)-(min)) | ((max)-(i)) )
9 #define _fore(i,a) for(int i=head[(a)];i;i=edge[i].nxt)
10 inline ll rr() {
11     ll s=0,w=1;char c=getchar();
12     while(_in(c,'0','9')<0) { if(c=='-') w*=-1; c=getchar(); }
13     while(_in(c,'0','9')>=0) { s=s*10+c-'0'; c=getchar(); }
14     return s*w;
15 }
16 inline void pr(ll x) { if(x>=10) pr(x/10); putchar(x%10+'0'); }
17 int main() {
18     printf("\n");
19     return 0;
20 }
```

1.2 预编译

头文件引入方式改为如下，可以把头文件放入 lab.h，然后使用 g++ lab.h 预编译。
实际编译使用 g++ lab.cpp -D YCYLOCAL 添加条件编译参数。

```
1 #ifdef YCYLOCAL
2 #include "lab.h"
3 #else // #include <bits/stdc++.h>
4 #include <list>          #include <stack>          #include <iostream>
5 #include <cmath>         #include <cstdint>         #include <algorithm>
6 #include <queue>         #include <cstring>         #include <functional>
7 #endif
```

1.3 进制转换

```

1 void pr_x(ll n,int x) {
2     char c = n%x; c += x>9?'A'-10:'0';
3     if(n>=x) pr(n/x,x); putchar(c);
4 }

```

1.4 常见技巧

向上取整 p/q 为 $(p-1)/q+1$ 。

1.5 快速幂

```

1 ll power(ll a,ll b,ll p) {
2     ll rst = 1%p;
3     for(;b>0;b>>=1,a=a*a%p)
4         if(b&1) rst=a*rst%p;
5     return rst;
6 }

```

1.6 矩阵快速幂

```

1 const int mod=1e9+7;
2 ll a[105][105],b,n;
3 ll ans[105][105]={0};
4 inline void jzcf1(){
5     ll c[105][105]={0}; //新矩阵放临时结果
6     _fora(k,1,n) _fora(i,1,n) _fora(j,1,n)
7         c[i][j]=(c[i][j]+ans[i][k]*a[k][j])%mod;
8     _fora(i,1,n) _fora(j,1,n)
9         ans[i][j]=c[i][j];
10 }
11 inline void jzcf2(){
12     ll c[105][105]={0};
13     _fora(k,1,n) _fora(i,1,n) _fora(j,1,n)
14         c[i][j]=(c[i][j]+ans[i][k]*a[k][j])%mod;
15     _fora(i,1,n) _fora(j,1,n)
16         a[i][j]=c[i][j];
17 }
18 int main(){
19     int n; //读入
20     _fora(i,1,n) ans[i][i]=1;

```

```

21     while(b){
22         if(b&1) jzcf1();//矩阵乘法
23         jzcf2(); b>=>1;
24     }
25 }

```

1.7 快速排序

```

1  ll nn[10010];
2  void q_sort(int l,int r) {
3      int i=l,j=r,x = nn[i];
4      while(i<j) {
5          while(nn[j]>x&&i<j) j--;
6          if(i<j) nn[i++] = nn[j];
7          while(nn[i]<x&&i<j) i++;
8          if(i<j) nn[j--] = nn[i];
9      } nn[i] = x;
10     if(l<r) { q_sort(l,i-1); q_sort(i+1,r); }
11 }

```

1.8 第 k 大数

```

1  ll nn[5000010],k,n;
2  ll q_sort(ll l,ll r) {
3      ll i=l,j=r,x=nn[(l+r)/2];
4      while(i<=j) {
5          while(nn[j]>x) j--;
6          while(nn[i]<x) i++;
7          if(i<=j) swap(nn[i++],nn[j--]);
8      } //l<=j<=i<=r
9      if(k<=j) return q_sort(l,j);
10     else if(k>=i) return q_sort(i,r);
11     else return nn[k+1];
12 }

```

1.9 链表

```

1  struct Node { int val; Node *prev,*next; };
2  struct List {
3      Node *head,*tail; int len;

```

```
4     List() {
5         head = new Node(); tail = new Node();
6         head->next = tail; tail->prev = head;
7         len = 0;
8     } // 在节点后 p 后插入值 v
9     void insert(Node *p,int v) {
10         Node *q = new Node(); q->val = v;
11         p->next->prev = q; q->next = p->next;
12         p->next = q; q->prev = p;
13         len++;
14     } // 删除节点 p
15     void erase(Node *p) {
16         p->prev->next = p->next; p->next->prev = p->prev;
17         delete p; len--;
18     } // 清空链表
19     ~List() {
20         while(head != tail) {
21             head = head->next;
22             delete head->prev;
23         }
24         delete tail; len = 0;
25     }
26 };
27 for(Node *p=l->head->next;p!=l->tail;p=p->next)
28     // 正序遍历
29 for(Node *p=l->tail->prev;p!=l->head;p=p->prev)
30     // 逆序遍历
```

第 2 章 数论

2.1 GCD 和 LCM

```
1 ll gcd(ll a, ll b) { return a?gcd(b%a,a):b; }
2 ll lcm(ll a, ll b) { return a/gcd(b%a,a)*b; }
3 ll gcd(ll a, ll b) { while(b) { ll t=a%b; a=b; b=t; } return a; }
```

2.2 EXGCD

对于方程 $ax + by = c$, 令 $g = \gcd(a, b)$, 若 $g \nmid c$ 则无解。

因此可通过 *exgcd* 先求出 $ax + by = g$ 的整数解, 再转换回原方程的解。

```
1 void exgcd(ll a, ll b, ll& x, ll& y) {
2     if(!b) { y=0; x=1; return; }
3     exgcd(b, a%b, y, x); y-=a/b*x;
4 }
```

2.3 Eratosthenes 筛

```
1 bool notn[100000001];
2 int prime[20000001], cnt;
3 void init(int n) {
4     _fora(i, 2, n) { if(!notn[i]) {
5         prime[++cnt] = i;
6         int tn = n/i;
7         _fora(j, i, tn) notn[i*j] = true;
8     } } return;
9 }
```

2.4 Euler 筛

```
1 bool notn[100000001];
2 int prime[20000001], cnt;
3 void init(int n){
4     _fora(i, 2, n) {
5         if(!notn[i]) prime[++cnt] = i;
6         int t = n/i;
7         _fora(j, 1, cnt) {
```

```

8         if(prime[j]>t) break;
9         notn[i*prime[j]] = true;
10        if(i%prime[j]==0) break;
11    }
12    } return;
13 }
```

2.5 素性测试

2.5.1 试除法

```

1 bool isprime(ll n) {
2     if(n<3) return n==2;
3     if(n&1==0) return false;
4     ll sn = (ll)sqrt(n*1.0);
5     for(ll i=3;i<=sn;i+=2)
6         if(n%i==0) return false;
7     return true;
8 }
```

2.5.2 Miller Rabbin

如果 $n \leq 2^{32}$, 那么 *ppp* 取 2, 7, 61; 如果 *ppp* 选择 2, 3, 7, 61, 24251, 那么 10^{16} 内只有唯一的例外。如果莫名 WA 了, 就多取点素数吧。

```

1 bool miller_rabbin(ll n) {
2     if (n<3) return n==2;
3     if(n&1==0) return false;
4     int a=n-1,b=0,j;
5     while (1-a&1) a/=2,++b;
6     int ppp[10] = {2,7,61};
7     _fora(i,0,2) {
8         int x = ppp[i];
9         if(n==x) return true;
10        ll v = power(x,a,n);
11        if(v==1||v==n-1) continue;
12        for(j=0;j<b;++j) {
13            v = v*v%n; if(v==n-1) break;
14        }
15        if(j>=b) return false;
16    } return true;
17 }
```

第 3 章 图论

3.1 链式前项星

```
1  const int MN = 10005; int head[MN];
2  struct Edge {int too,nxt,len;} edge[MN*2];
3  void add(int frm,int too,int len) {
4      static int cnt = 0;
5      edge[++cnt] = { too,head[frm],len };
6      head[frm] = cnt;
7  }
8  void dfs(int x,int fa) {
9      _fore(i,x) if(edge[i].too!=fa)
10         dfs(edge[i].too,x);
11 }
```

3.2 Dijkstra

```
1  int dis[MN];
2  struct Dis {
3      int dis,pos;
4      bool operator <(const Dis& x) const
5          { return x.dis<dis; }
6  };
7  void dijkstra(int ss) {
8      memset(dis,0x3f,sizeof(dis));
9      dis[ss] = 0;
10     priority_queue<Dis> pq; pq.push({0,ss});
11     while(!pq.empty()) {
12         Dis td = pq.top(); pq.pop();
13         int d=td.dis, x=td.pos;
14         if(d!=dis[x]) continue;
15         _fore(i,x) {
16             int y=edge[i].too, z=dis[x]+edge[i].len;
17             if(dis[y]>z) dis[y]=z, pq.push({dis[y],y});
18         }
19     }
20 }
```

3.3 Bellman-Ford

```

1 void bellman_ford(int ss) {
2     memset(dis,0x3f,sizeof(dis)); dis[ss]=0;
3     _fora(iaa,1,n-1) { int flag=1;
4         _fora(i,1,n) { int size=ee[i].size();
5             _fora(j,0,size-1) {
6                 int v=ee[i][j].nxt, t=dis[i]+ee[i][j].len;
7                 if(dis[ee[i][j].nxt]>t) {
8                     dis[ee[i][j].nxt] = t;
9                     flag = 0;
10                }
11            }
12        } if(flag) return;
13    }
14 }
```

3.4 Floyd

起始条件 $f(i,j) = \text{edge}(i,j)$, $f(i,i) = 0$ 。

```

1 inline void floyd() {
2     _fora(k,1,n) { _fora(i,1,n) {
3         if(i==k||f[i][k]==0x3f3f3f3f) continue;
4         _fora(j,1,n) f[i][j] = min(f[i][j],f[i][k]+f[k][j]);
5     } }
6 }
```

3.5 最近公共祖先 (LCA)

如果数据小，可以不用求 \log_2 ，直接莽 20。

```

1 int fa[MN][30],lgb[MN],depth[MN];
2 void lca_dfs(int now,int fat) {
3     _fora(i,1,20) lgb[i]=lgb[i>1]+1; lgb[1]=0;
4     fa[now][0] = fat;
5     depth[now] = depth[fat]+1;
6     _fora(i,1,lgb[depth[now]])
7         fa[now][i] = fa[fa[now][i-1]][i-1];
8     _fore(i,now) if(edge[i].too!=fat)
9         lca_dfs(edge[i].too,now);
10 }
```

```
11 int lca(int x,int y) {
12     if(depth[x]<depth[y]) swap(x,y);
13     while(depth[x]>depth[y])
14         x = fa[x][lgb[depth[x]-depth[y]]];
15     if(x==y) return x;
16     _forz(k,lgb[depth[x]]-1,0)
17         if(fa[x][k]!=fa[y][k])
18             { x=fa[x][k]; y=fa[y][k]; }
19     return fa[x][0];
20 }
```

第 4 章 动态规划

4.1 背包

4.1.1 01 背包

给定体积为 v_i ，价值 w_i 的 N 个物品，背包容积为 M ，每个物品只能取 1 个，求最大价值。

```
1 _fora(i,1,n) _forz(j,m,v[i])
2     dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
3 _fora(j,1,m) ans = max(ans,dp[j]);
```

4.1.2 完全背包

给定体积为 v_i ，价值 w_i 的 n 个物品，背包容积为 v ，每个物品任意取，求最大价值。

```
1 _fora(i,1,n) _fora(j,v[i],m)
2     dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
3 _fora(j,1,m) ans = max(ans,dp[j]);
```

4.1.3 多重背包

给定体积为 v_i ，价值 w_i 的 N 个物品，背包容积为 M ，每个物品有 c_i 个，求最大价值。

如各种背包组合（如洛谷 P1833 樱花），通常把完全背包转为 99999 个（适当调节）多重背包，再按 01 背包来。

```
1 int tm=1,vv[],ww[];
2 _fora(i,1,n) {
3     int tc = c[i];
4     for(int b=1;b<p;b<=1,tc-=b,++tm) {
5         vv[tm] = v[i]*b;
6         ww[tm] = w[i]*b;
7     }
8     vv[tm] = v[i]*tc;
9     ww[tm] = w[i]*tc;
10    ++tm;
11 }
12 _fora(i,1,n) _forz(j,m,vv[i])
13     dp[j]=max(dp[j],dp[j-vv[i]]+ww[i]);
14 _fora(j,1,m) ans = max(ans,dp[j]);
```

4.2 最长公共上升序列

给出 $1, 2, \dots, n$ 的两个排列 a 和 b ，求它们的最长公共子序列。

```

1  int f[MN],ma[MN],b[MN];
2  int n,len=0; memset(f,0x3f,sizeof(f)); f[0]=0;
3  _fora(i,1,n) { ma[rr()]=i; } _fora(i,1,n) { b[i]=rr(); }
4  _fora(i,1,n) {
5      int l=0,r=len;
6      if(ma[b[i]]>f[len]) f[++len]=ma[b[i]];
7      else { while(l<r) {
8          int mid=(l+r)/2;
9          if(f[mid]>ma[b[i]])r=mid;
10         else l=mid+1;
11     } } f[l]=min(ma[b[i]],f[l]);
12 }
```

4.3 数字计数

试计算在区间 1 到 n 的所有整数中，数字 $x(0 \leq x \leq 9)$ 共出现了多少次？

```

1  ll addup(ll n, ll x, ll k, ll l){ //k=1,l=0
2      ll a=0; if(n>9) a=addup(n/10,x,k*10,l+1);
3      n%=10; sum += ((n>x)+n*a)*k+n*l*k/10-(!x)*k;
4      return a+(n==x);
5  }
```
