

ACM模板精简版本

by 鱼竿钓鱼干

E-mail:851892190@qq.com

参考：acwing板子（以这个为主+个人做题经验补充），洛谷题解，各类博客，各平台比赛，算法竞赛进阶指南

版本：2021/1/30

更新内容：删除没必要的东西，添加一些DP和图论板子，对差分edit增加鲁棒性,试除法优化

ACM模板精简版本

做题板板（别抄，这个原模原样copy我的话会被判重的，自重）

双指针

数学

质数

试除法优化

线性筛

质因数

分解质因数

试除法求约数

约数个数(多个数相乘的)

约数个数和

约数和

欧拉函数

筛法求欧拉函数（1~n,欧拉函数之和）

GCD/LCM

GCD

阶乘

阶乘位数

进制与位运算

进制转换

10进制转K进制

K进制转10进制

Tip

二进制枚举

枚举子集 $O(2^{|S|})$, $|S|$ 表示集合中元素个数

枚举方案

枚举子集的子集 $O(3^n)$

位运算

求二进制中1的个数

$\log_2(x)$ 取整

求 $\log_2(x)$ （暴力） $O(\log x)$

求 $\log_2(x)$ （二分） $O(\log \log x)$

预处理 $\log_2(x)$ 1~n中所有数 $O(n)$

满足方程 $x^k \leq y$ 的最大的k

高精

A+B

A-B

A*b

A/b

大数阶乘（vector太慢了，用数组）

- 其他处理
- 去前导0
- 多组输入初始化
- 排序
 - 归并
 - 排序
 - 求逆序对
 - 逆序对应用
- 数据结构
 - STL
 - 栈
 - 表达式
 - 后缀表达式
 - 单调栈
 - 队列
 - 普通队列
 - 单调队列(滑动窗口)
 - Trie树
 - 存储查找字符串集合
 - 并查集
 - 普通并查集
 - 维护距离(向量本质, 有向图)
 - 维护大小
 - 扩展域
 - 链表
 - 普通链表
 - 双链表
 - 哈希表
 - 堆
- 查找
 - 二分查找
 - 二分的应用
 - 整数二分
 - 浮点数二分
- 搜索与图论
 - 图
 - 图的存储
 - 邻接表
 - 邻接矩阵
 - 建图小技巧
 - DFS
 - 有多少个连通块(洪泛)
 - DFS遍历图(树的重心)
 - 列出所有解
 - 全排列
 - STL
 - 组合输出
 - BFS
 - 最短步数(边的权值均为1, STL写法)
 - BFS+路径保存
 - BFS遍历图(边权1最短路, 手动模拟队列写法)
 - 连通块里多少块
 - 拓扑序列(有向无环图AOV)
- 最短路
 - dijkstra朴素稠密图 $O(n^2)$
 - dijkstra堆优化稀疏图 $O(m\log n)$
 - dijkstra反向建图求多个点到起点的最短路
 - bellman_ford()(处理边数限制)

- SPFA
- SPFA (判有无负权环)
- Floyd(多源汇最短路)
- Floyd求最短环
- 最小生成树
 - Kruskal (稀疏图) ($O(m\log m)$)
 - Prim(稠密图)($O(n^2)$)
- 二分图
 - 染色法判二分图($O(m+n)$)
 - 匈牙利算法二分图最大匹配图
- 前缀和/差分
 - 一维前缀和
 - 二维前缀和
 - Tip:前缀和和一些注意点(激光炸弹为例)
 - 一维差分
 - 二维差分
 - 注意事项
- 字符串
 - KMP
- 区间操作
 - 区间合并
- DP
 - DP思考方式
 - 状态表示
 - 集合
 - 属性
 - 状态计算
 - 划分
 - 计算过程
 - 背包
 - 01背包
 - 二维
 - 一维
 - 完全背包
 - 二维
 - 二维优化
 - 一维优化
 - 多重背包
 - 暴力朴素
 - 二进制优化
 - 分组背包
 - 背包方案数
 - 二维
 - 一维
 - 线性DP
 - 数字三角
 - 最长上升子序列 (LIS)
 - 朴素DP $O(N^2)$
 - 记录最大上升子序列
 - 优化 $O(N\log N)$
 - 最长公共子序列 (LCS)
 - 区间DP
 - 记忆化搜索

做题板板（别抄，这个原模原样copy我的话会被判重的，自重）

```
1 //Author fishingrod
2 #pragma GCC optimize(2)
3 #pragma GCC optimize(3)
4 #pragma GCC optimize("Ofast")
5 #include<bits/stdc++.h>
6 using namespace std;
7 #define ll long long
8 #define int long long
9 #define ull unsigned long long
10 #define pii pair<int,int>
11 #define db double
12 #define pb push_back
13 #define dbg cout<<"ok"<<endl;
14 #define mst(A) memset(A,0,sizeof A);
15 #define tip cout<<"problem,date,test,calm!"<<endl;
16 #define prn(x) printf("%lld\n",x);
17 #define pr printf
18 #define sc scanf
19 #define endl "\n"
20 #define rep(i,a,b) for(ll i=a,i<=(b);++i)
21 #define urep(i,a,b) for(ll i=a;i>=b;--i)
22 const int N=1e5+10,INF=0x3f3f3f3f,Mod=998244353;
23 const db eps=1e-7,pi=acos(-1.0);;
24
25 ll read()
26 {
27     ll x=0,f=0;char ch=getchar();
28     while(!isdigit(ch))f|=ch=='-',ch=getchar();
29     while(isdigit(ch))x=10*x+ch-'0',ch=getchar();
30     return f?-x:x;
31 }
32
33 ll fp(ll a,ll b,ll mod)
34 {
35     ll res=1;a=a%mod;
36     while(b)
37     {
38         if(b&1)res=(res*a)%mod;
39         a=(a*a)%mod;b>>=1;
40     }
41     return res;
42 }
43 /*
44 bool check1(ll mid){}
45 bool check2(ll mid){}
46 ll bs1(ll l,ll r)
47 {
48     while(l<r)
49     {
50         ll mid=l+r>>1;
51         if(check1(mid))r=mid;
52         else l=mid+1;
53     }
54     return l;
55 }
```

```

56 | ll bs2(ll l,ll r)
57 | {
58 |     while(l<r)
59 |     {
60 |         ll mid=l+r+1>>1;
61 |         if(check2(mid))l=mid;
62 |         else r=mid-1;
63 |     }
64 |     return l;
65 | }
66 | */
67 | int T,Q,n,m,k,p,ans,cnt,sum,tmp;
68 |
69 | signed main()
70 | {
71 |     tip;
72 |
73 |     return 0;
74 | }

```

双指针

```

1 | for(int i = 0, j = 0; i < n; i ++ )//考虑起点，都为开头/两端什么的
2 | {
3 |     //check函数一般反着写，while循环一直缩小边界，直到可能为答案的区间
4 |     //注意是可能，比如上面一些题目<和=就是可能的情况，但是更新答案只在=的时候
5 |     while (j < i && check(i, j)) j ++ ;//注意边界问题
6 |     //更新答案
7 | }

```

数学

质数

试除法优化

$O(\sqrt{N}/3)$

```

1 | bool is_prime(ll a)
2 | {
3 |     if(a==1)return 0;
4 |     if(a==2||a==3)return 1;
5 |     if(a%6!=1&&a%6!=5)return 0;
6 |     for(int i=5;i<=a/i;i+=6)
7 |         if(a%i==0||a%(i+2)==0)return 0;
8 |     return 1;
9 | }

```

线性筛

```
1  int primes[N], cnt;    // primes[] 存储所有素数
2  bool st[N];           // st[x] 存储x是否被筛掉，后面可以直接用来判是否为素数
3  //n只会被最小质因子筛掉
4  void get_primes(int n)
5  {
6      memset(st, 0, sizeof st);
7      st[0]=st[1]=1;
8      for (int i = 2; i <= n; i ++ )//不要忘记等号
9      {
10         if (!st[i]) primes[cnt++] = i;
11         for (int j = 1; primes[j] <= n / i; j ++ )//不要忘记等号
12         {
13             st[primes[j] * i] = true;//合数一定有最小质因数，用最小质因数的倍数筛去
14             //合数
15             if (i % primes[j] == 0) break;//prime[j]一定是i最小质因子，也一定是
16             //prime[j]*i的最小质因子
17         }
18     }
```

质因数

分解质因数

$O(\sqrt{N})$

```
1  void divide(int n)
2  {
3      for(int i=2;i<n/i;i++)//不要忘记等号
4          if(n%i==0)//i一定是质数
5          {
6              int s=0;
7              while(n%i==0)
8              {
9                  n/=i;
10                 s++;
11             }
12             printf("%d %d\n", i, s);
13         }
14         if(n>1)printf("%d %d\n", n, 1);//处理唯一一个>sqrt(n)的
15         puts(" ");
16     }
17     /*
18     给定两个数n, m, 其中m是一个素数。
19     将n (0<=n<=2^31) 的阶乘分解质因数，求其中有多少个m。
20     while(n/m) ans+=n/m, n/=m;
21     */
```

试除法求约数

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 vector<int>get_divisors(int n)
5 {
6     vector<int>res;
7     for(int i=1;i<=n/i;i++)//从1开始，约数啊
8         if(n%i==0)
9         {
10             res.push_back(i);
11             if(i!=n/i)res.push_back(n/i);//约数通常成对出现，特判完全平方
12         }
13     sort(res.begin(),res.end());
14     return res;
15 }
16
17 int main()
18 {
19     int n;
20     cin>>n;
21     while(n-->0)
22     {
23         int x;
24         cin>>x;
25         auto res=get_divisors(x);
26         for(auto t:res)cout<<t<<' ';
27         cout<<endl;
28     }
29 }
```

约数个数(多个数相乘的)

$O(N\sqrt{N})$ 可以先线性筛预处理优化

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 typedef long long LL;
5
6 const int mod=1e9+7;
7 int main()
8 {
9     int n;
10    cin>>n;
11    unordered_map<int,int>primes;
12    while(n-->0)
13    {
14        int x;
15        cin>>x;
16        for(int i=2;i<=x/i;i++)
17            while(x%i==0)
18            {
19                x/=i;
20                primes[i]++;
21            }
22    }
```

```

22         if(x>1)primes[x]++;
23     }
24     LL res=1;
25     for(auto prime:primes)res=res*(prime.second+1)%mod;
26     cout<<res<<endl;
27     return 0;
28 }

```

约数个数和

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  int main()
6  {
7      int res=0,n;
8      cin>>n;
9      for(int i=1;i<=n;i++)res+=n/i;
10     cout<<res;
11     return 0;
12 }

```

约数和

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int mod=1e9+7;
5  typedef long long LL;
6  int main()
7  {
8      int n,x;
9      unordered_map<int,int>primes;
10     cin>>n;
11     while(n-->0)
12     {
13         cin>>x;
14         for(int i=2;i<=x/i;i++)
15             while(x%i==0)
16             {
17                 x/=i;
18                 primes[i]++;
19             }
20         if(x>1)primes[x]++;
21     }
22     LL res=1;
23     for(auto prime:primes)
24     {
25         int p=prime.first,a=prime.second;
26         LL t=1;
27         while(a-->0)t=(t*p+1)%mod;
28         res=res*t%mod;
29     }
30     cout<<res<<endl;
31     return 0;
32 }

```


欧拉函数

```
1  ll phi(ll x)
2  {
3      ll res = x;
4      for (int i = 2; i <= x / i; i ++ )
5          if (x % i == 0)
6          {
7              res = res / i * (i - 1);
8              while (x % i == 0) x /= i;
9          }
10     if (x > 1) res = res / x * (x - 1);
11
12     return res;
13 }
```

筛法求欧拉函数 (1~n,欧拉函数之和)

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5  const int N=1e6+10;
6  ll primes[N],n,phi[N],cnt;
7  bool st[N];
8
9  ll get_eulers(ll n)
10 {
11     phi[1]=1;
12     for(int i=2;i<=n;i++)
13     {
14         if(!st[i])
15         {
16             primes[cnt++]=i;
17             phi[i]=i-1;
18         }
19         for(int j=0;primes[j]<=n/i;j++)
20         {
21             st[i*primes[j]]=1;
22             if(i%primes[j]==0)
23             {
24                 phi[i*primes[j]]=phi[i]*primes[j];
25                 break;
26             }
27             phi[i*primes[j]]=phi[i]*(primes[j]-1);
28         }
29     }
30     ll res=0;
31     for(int i=1;i<=n;i++)res+=phi[i];
32     return res;
33 }
34
35 int main()
36 {
37     int n;
38     cin>>n;
```

```

39 |     cout<<get_eulers(n);
40 |
41 | }

```

GCD/LCM

GCD

```

1 | ll gcd(ll a, ll b)
2 | {
3 |     return b ? gcd(b, a % b) : a;
4 | }

```

阶乘

阶乘位数

```

1 | #include<bits/stdc++.h>
2 | using namespace std;
3 | int main()
4 | {
5 |     double n;
6 |     while(cin>>n)
7 |     {
8 |         double s=0;
9 |         for(int i=1;i<=n;i++)s+=log10(i);//n!取对数
10 |         cout<<int(s)+1<<endl;
11 |     }
12 |     return 0;
13 | }

```

进制与位运算

进制转换

10进制转K进制

```

1 | string itoA(LL n,int radix)    //n是待转数字，radix是指定的进制
2 | {
3 |     string ans;
4 |     do{
5 |         int t=n%radix;
6 |         if(t>=0&&t<=9) ans+=t+'0';
7 |         else ans+=t-10+'A';//注意大小写
8 |         n/=radix;
9 |     }while(n!=0);    //使用do{}while（）以防止输入为0的情况
10 |     reverse(ans.begin(),ans.end());//逆序翻转
11 |     return ans;
12 | }

```

K进制转10进制

```
1 LL Atoi(string s,int radix)    //s是给定的radix进制字符串
2 {
3     LL ans=0;
4     for(int i=0;i<s.size();i++)
5     {
6         char t=s[i];
7         if(t>='0'&&t<='9') ans=ans*radix+t-'0';
8         else ans=ans*radix+t-'A'+10;
9     }
10    return ans;
11 }
```

Tip

1. 有时候不用真的转换，可以直接按格式打印

二进制枚举

枚举子集 $O(2^{|S|})$, $|S|$ 表示集合中元素个数

```
1 // s 是一个二进制数（二进制每位数字对应一种状态），表示一个集合，i 枚举 s 的所有子集
2 //s=9=1001,有子集9=1001,8=1000,1=0001
3 for (int i = s; i; i = s & i - 1)
4 // [0, 2^n-1], n个元素的子集有2^n个
5 for(int i = 0; i < (1<<n); i++)
```

枚举方案

```
1 void binary_enum(int n)
2 {
3     for(int i=0;i<(1<<n);i++)
4         for(int j=0;j<n;j++)
5         {
6             if(i&(1<<j))
7             {
8                 array[j]//别写成array[i]
9             }
10        }
11 }
```

枚举子集的子集 $O(3^n)$

```
1 for (int s = 0; s < 1 << n; s ++ ) // 枚举集合 {0, ..., n - 1} 的所有子集
2     for (int i = s; i; i = s & i - 1) // 枚举子集 s 的子集
3         // blablabla
```

位运算

```
1  a&b按位与
2  a|b按位或
3  a^b按位异或//可以用来排除出现偶数次的数
4  ~a按位取反, 用于表示负数 -x = ~x + 1
5  a<<b = a * 2^b
6  a>>b = a / 2^b (去小数)
7  !a非
8
9  将 x 第 i 位取反: x ^= 1 << i
10 将 x 第 i 位制成 1: x |= 1 << i
11将 x 第 i 位制成 0: x &= ~1 << i 或 x &= ~(1 << i)
12取 x 对 2 取模的结果: x & 1
13取 x 的第 i 位是否为 1: x & 1 << i 或 x >> i & 1
14取 x 的最后一位: x & -x
15取 x 的绝对值: (x ^ x >> 31) - (x >> 31) (int 型)
16判断 x 是否不为 2 的整次方幂: x & x - 1
17判断 a 是否不等于 b: a != b, a - b, a ^ b
18判断 x 是否不等于 -1: x != -1, x ^ -1, x + 1, ~x
19
20异或, 加法恒等式
21a+b=a|b+2(a&b) 若a+b=a|b-->a&b=0
22若a-b=a|b-->a-b每一位不能发生借位, 即若a已知, 对于a的每一位, 若为1, 则可以01若为0, 则指南0
```

求二进制中1的个数

```
1  bitset(sizeof(int)*8)b(x);
2  cout<<b.count()<<endl;
```

log2(x)取整

求log₂(x) (暴力) O(logx)

```
1  int log_2(int x)
2  {
3      int res = 0;
4      while (x >> 1) res ++, x >>= 1;
5      return res;
6  }
```

求log₂(x) (二分) O(loglogx)

```
1  int log_2(int x)
2  {
3      int l = 0, r = 31, mid;
4      while (l < r)
5      {
6          mid = l + r + 1 >> 1;
7          if (x >> mid) l = mid;
8          else r = mid - 1;
9      }
10 }
```

预处理log₂(x) 1~n中所有数 O (n)

```
1 int log_2[N]; // 存 log2(i) 的取整结果
2 void init(int n)
3 {
4     for (int i = 0; 1 << i <= n; i++)
5         log_2[1 << i] = i;
6     for (int i = 1; i <= n; i++)
7         if (!log_2[i]) log_2[i] = log_2[i - 1];
8 }
```

满足方程 $x^k \leq y$ 的最大的k

```
1 ll calc(ll x, ll y) // x, y, k 整数
2 {
3     if (x <= 1 || y == 0) return -1; // k 不存在或无限大
4     ll ans = 0;
5     while (y >= x) ans++, y /= x; // 不用想 log 什么的
6     return ans;
7 }
```

高精

A+B

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 // 如果k进制, 那么10都改成k就行了, 传进去的时候注意改A~10, F~15;
4 vector<int> add(vector<int> &A, vector<int> &B)
5 {
6     if (A.size() < B.size()) return add(B, A);
7     vector<int> c;
8     int t = 0; // 一定要初始化为0
9     for (int i = 0; i < A.size(); i++) // A+B+t
10     {
11         t += A[i];
12         if (i < B.size()) t += B[i];
13         c.push_back(t % 10);
14         t /= 10;
15     }
16     if (t) c.push_back(t); // 处理最高位
17     return c;
18 }
19 int main()
20 {
21     string a, b;
22     vector<int> A, B;
23     cin >> a >> b;
24     for (int i = a.size() - 1; i >= 0; i--) A.push_back(a[i] - '0'); // 逆序输入, 方便进位
25     for (int i = b.size() - 1; i >= 0; i--) B.push_back(b[i] - '0');
26     auto c = add(A, B);
27     for (int i = c.size() - 1; i >= 0; i--) cout << c[i]; // 逆序输出
28     return 0;
29 }
```

A-B

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  void trimzero(vector<int> &A)//处理输入前的0和输出时的0
6  {
7      while(A.size()>1&&A.back()==0)A.pop_back();
8  }
9  bool cmp(vector<int> &A,vector<int> &B)
10 {
11     if(A.size()!=B.size())return A.size()>B.size();
12     for(int i=A.size()-1;i>=0;i--)
13         if(A[i]!=B[i])return A[i]>B[i];
14     return 1;
15 }
16
17
18 vector<int>sub(vector<int> &A,vector<int> &B)
19 {
20     vector<int>c;
21     for(int i=0,t=0;i<A.size();i++)
22     {
23         t=A[i]-t;
24         if(i<B.size())t-=B[i];
25         c.push_back((t+10)%10);
26         if(t<0)t=1;
27         else t=0;
28     }
29     trimzero(c);
30     return c;
31 }
32
33
34 int main()
35 {
36     string a,b;
37     cin>>a>>b;
38     vector<int>A,B;
39     for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
40     for(int i=b.size()-1;i>=0;i--)B.push_back(b[i]-'0');
41     trimzero(A);
42     trimzero(B);
43     if(cmp(A,B))
44     {
45         auto c=sub(A,B);
46         for(int i=c.size()-1;i>=0;i--)cout<<c[i];
47     }
48     else
49     {
50         auto c=sub(B,A);
51         cout<<"-";
52         for(int i=c.size()-1;i>=0;i--)cout<<c[i];
53     }
54     return 0;
55 }
```

A*b

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  void trimzero(vector<int> &A)
6  {
7      while(A.size()>1&&A.back()==0)A.pop_back();
8  }
9  vector<int> mul(vector<int> &A,int b)
10 {
11     vector<int>c;
12     for(int i=0,t=0;i<A.size()||t;i++)
13     {
14         if(i<A.size())t+=A[i]*b;
15         c.push_back(t%10);
16         t/=10;
17     }
18     trimzero(c);
19     return c;
20 }
21 int main()
22 {
23     string a;
24     int b;
25     cin>>a>>b;
26     vector<int>A;
27     for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
28     trimzero(A);
29     auto c=mul(A,b);
30     for(int i=c.size()-1;i>=0;i--)cout<<c[i];
31     return 0;
32 }
```

A/b

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  void trimzero(vector<int> &A)
6  {
7      while(A.size()>0&&A.back()==0)A.pop_back();
8  }
9  vector<int>div(vector<int> &A,int b,int &r)
10 {
11     vector<int>c;
12     r=0;
13     for(int i=A.size()-1;i>=0;i--)//出发比较特别从高位开始搞
14     {
15         r=r*10+A[i];
16         c.push_back(r/b);
17         r%=b;
18     }
19 }
```

```

20     reverse(c.begin(),c.end());
21     trimzero(c);
22     return c;
23 }
24 int main()
25 {
26     string a;
27     int b;
28     vector<int>A;
29     cin>>a>>b;
30     for(int i=a.size()-1;i>=0;i--)A.push_back(a[i]-'0');
31     int r;
32     auto c=div(A,b,r);
33     for(int i=c.size()-1;i>=0;i--)cout<<c[i];
34     cout<<endl<<r<<endl;
35     return 0;
36 }

```

大数阶乘 (vector太慢了, 用数组)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6      int n,ws;
7      while(scanf("%d",&n)!=EOF)
8      {
9          double s=0;
10         for(int i=1;i<=n;i++)s+=log10(i);
11         ws=int(s)+1;//求位数
12         int f[ws];
13         memset(f,0,sizeof(f));
14         int ans,jw,j;
15         f[0]=1;
16         for(int i=2;i<=n;i++)
17         {
18             int jw=0;
19             for(j=0;j<ws;j++)
20             {
21                 int ans=f[j]*i+jw;
22                 f[j]=ans%10;
23                 jw=ans/10;
24             }
25         }
26         for(j=ws-1;j>=0;j--)printf("%d",f[j]);
27         printf("\n");
28     }
29     return 0;
30 }

```


其他处理

去前导0

```
1 void trimzero(vector<int> &A)//处理输入前的0和输出时的0
2 {
3     while(A.size()>1&&A.back()==0)A.pop_back();
4 }
```

多组输入初始化

```
1 多次输入或者累计运算
2 记得清空vector
3 vector<int>a;
4 a.clear();
```

排序

归并

排序

```
1 int q[N],tmp[N];
2 void merge_sort(int q[],int l,int r)//这里只有<=>没有</>
3 {
4     if(l>=r)return;
5     int mid=l+r>>1;
6     merge_sort(q,l,mid),merge_sort(q,mid+1,r);
7     int k=0,i=l,j=mid+1;
8     while(i<=mid&&j<=r)
9     {
10         if(q[i]<=q[j])tmp[k++]=q[i++];
11         else tmp[k++]=q[j++];
12     }
13     while(i<=mid)tmp[k++]=q[i++];
14     while(j<=r)tmp[k++]=q[j++];
15
16     for(int i=l,j=0;i<=r;i++,j++)q[i]=tmp[j];//不要写成i=1
17 }
18 }
```

求逆序对

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4
5 typedef long long ll;
6 const int N=1e6+10;
7
8
9 int n;
10 int q[N],tmp[N];
11
```

```

12 //i不会等于1只有1
13 ll merge_sort(int l,int r)
14 {
15     if(l>=r)return 0;
16     int mid=l+r>>1;
17     ll res=merge_sort(l,mid)+merge_sort(mid+1,r);
18     int k=0,i=l,j=mid+1;//i是l别打成1
19     while(i<=mid&& j<=r)
20     {
21         if(q[i]<=q[j])tmp[k++]=q[i++];
22         else
23         {
24             tmp[k++]=q[j++];
25             res+=mid-i+1;//q[i]>q[j], 左区间剩下的所有数与右区间当前数成为逆序对
26         }
27     }
28     while(i<=mid)tmp[k++]=q[i++];    //扫尾
29     while(j<=r)tmp[k++]=q[j++];
30     for(int i=l,j=0;i<=r;i++,j++)q[i]=tmp[j];//不要写成i=1
31     return res;
32 }
33
34
35 int main()
36 {
37     int n;
38     cin>>n;
39     for(int i=0;i<n;i++)cin>>q[i];
40     cout<<merge_sort(0,n-1);
41 }

```

逆序对应用

1. 交换重排，根据奇偶性判局面可达（可能是字符串，二维平面，数组序列）

数据结构

STL

```

1  vector, 变长数组, 倍增的思想
2      size()  返回元素个数
3      empty()  返回是否为空
4      clear()  清空
5      front()/back()
6      push_back()/pop_back()
7      begin()/end()
8      []
9      支持比较运算, 按字典序
10
11
12  pair<int, int>
13      first, 第一个元素
14      second, 第二个元素
15      支持比较运算, 以first为第一关键字, 以second为第二关键字（字典序）
16
17

```

```

18 string, 字符串
19     size()/length()  返回字符串长度
20     empty()
21     clear()
22     substr(起始下标, (子串长度))  返回子串
23     c_str()  返回字符串所在字符数组的起始地址
24
25
26 queue, 队列
27     size()
28     empty()
29     push()  向队尾插入一个元素
30     front()  返回队头元素
31     back()  返回队尾元素
32     pop()  弹出队头元素
33
34
35 priority_queue, 优先队列, 默认是大根堆
36     size()
37     empty()
38     push()  插入一个元素
39     top()  返回堆顶元素
40     pop()  弹出堆顶元素
41     定义成小根堆的方式: priority_queue<int, vector<int>, greater<int>> q;
42
43
44 stack, 栈
45     size()
46     empty()
47     push()  向栈顶插入一个元素
48     top()  返回栈顶元素
49     pop()  弹出栈顶元素
50
51
52 deque, 双端队列
53     size()
54     empty()
55     clear()
56     front()/back()
57     push_back()/pop_back()
58     push_front()/pop_front()
59     begin()/end()
60     []
61
62
63 set, map, multiset, multimap, 基于平衡二叉树(红黑树), 动态维护有序序列
64     size()
65     empty()
66     clear()
67     begin()/end()
68     ++, -- 返回前驱和后继, 时间复杂度  $O(\log n)$ 
69
70
71 set/multiset
72     insert()  插入一个数
73     find()  查找一个数
74     count()  返回某一个数的个数(由于set不重复原则所以只返回01)
75     erase()

```

```

76         (1) 输入是一个数x, 删除所有x     $O(k + \log n)$ 
77         (2) 输入一个迭代器, 删除这个迭代器
78         lower_bound()/upper_bound()
79         lower_bound(x)  返回大于等于x的最小的数的迭代器
80         upper_bound(x)  返回大于x的最小的数的迭代器
81     map/multimap
82         insert()  插入的数是一个pair
83         erase()   输入的参数是pair或者迭代器
84         find()
85         []  注意multimap不支持此操作。 时间复杂度是  $O(\log n)$ 
86         lower_bound()/upper_bound()
87
88
89     unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
90     和上面类似, 增删改查的时间复杂度是  $O(1)$ 
91     不支持 lower_bound()/upper_bound(),  迭代器的++, --
92
93
94     bitset, 压位
95         bitset<10000> s;
96         ~, &, |, ^
97         >>, <<
98         ==, !=
99         []
100
101
102         count()  返回有多少个1
103
104
105         any()  判断是否至少有一个1
106         none()  判断是否全为0
107
108
109         set()  把所有位置成1
110         set(k, v)  将第k位变成v
111         reset()  把所有位变成0
112         flip()  等价于~
113         flip(k)  把第k位取反
114
115

```

栈

表达式

后缀表达式

单调栈

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=1e6+5;
5
6  int n;
7  int stk[N],tt;

```

```

8
9
10 int main()
11 {
12     cin>>n;
13     for(int i=0;i<n;i++)
14     {
15         int x;
16         cin>>x;
17         while(tt&&stk[tt]>=x)tt--;//1~i-1单调递增的栈,出栈了就不会再回来了
18         if(tt)cout<<stk[tt]<<" ";
19         else cout<<-1<<" ";
20         stk[++tt]=x;
21     }
22 }

```

队列

普通队列

```

1 // hh 表示队头, tt表示队尾, 队列从0开始
2 // 注意关注N的大小, 很有可能越界, 如果多次入队, 推荐用queue
3 int q[N], hh = 0, tt = -1;
4 // 如果第一个要插入队尾的元素已经知道了, 那tt开局用0即可
5 int hh=0, tt=-1
6 q[0]=1;
7 // 向队尾插入一个数
8 q[ ++ tt] = x;
9
10 // 从队头弹出一个数
11 hh ++ ;
12
13 // 队头的值
14 q[hh];
15 int t=q[hh++]; // t变为队头同时弹出原来的队头常用于bfs
16
17 // 判断队列是否为空
18 if (hh <= tt)
19 {
20
21 }
22
23 // 循环
24 q[++tt]=q[hh];
25 hh++;

```

单调队列(滑动窗口)

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4
5 const int N=1e6+5;
6
7

```

```

8   int a[N],q[N],n,k;//q存下标
9
10
11  int main()
12  {
13      scanf("%d%d",&n,&k);
14      for(int i=0;i<n;i++)scanf("%d",&a[i]);
15      int hh=0,tt=-1;
16      for(int i=0;i<n;i++)
17      {
18          //判断队头是已经滑出窗口
19          if(hh<=tt&&i-k+1>q[hh])hh++;
20          while(hh<=tt&&a[q[tt]]>=a[i])tt--;
21          q[++tt]=i;
22          if(i>=k-1)printf("%d ",a[q[hh]]);
23      }
24      puts("");
25      hh=0,tt=-1;
26      memset(q,0,sizeof(q));
27      for(int i=0;i<n;i++)
28      {
29          if(hh<=tt&&i-k+1>q[hh])hh++;
30          while(hh<=tt&&a[q[tt]]<=a[i])tt--;
31          q[++tt]=i;
32          if(i>=k-1)printf("%d ",a[q[hh]]);
33      }
34      return 0;
35  }
36

```

Trie树

存储查找字符串集合

```

1   #include<bits/stdc++.h>
2   using namespace std;
3
4
5   const int N=1e6+10;
6
7
8   int son[N][26],cnt[N],idx;//下标是0的点既是根节点又是空节点，cnt是对应每个停止符的数量。
9   char str[N];
10
11
12  void insert(char *str)
13  {
14      int p=0;
15      for(int i=0;str[i];i++)
16      {
17          int u=str[i]-'a';
18          if(!son[p][u])son[p][u]=++idx;//p是根u是儿子，如果没有儿子，idx只是查询有
          没有
19          p=son[p][u];
20      }
21      cnt[p]++;

```

```

22 }
23
24
25 int query(char *str)
26 {
27     int p=0;
28     for(int i=0;str[i];i++)
29     {
30         int u=str[i]-'a';
31         if(!son[p][u])return 0;
32         p=son[p][u];
33     }
34     return cnt[p];
35 }
36 int main()
37 {
38     int n;
39     char op[2];
40     scanf("%d",&n);
41     while(n--)
42     {
43         scanf("%s%s",op,str);
44         if(op[0]=='I')insert(str);
45         else printf("%d\n",query(str));
46     }
47     return 0;
48 }

```

并查集

普通并查集

```

1  int find(int x)
2  {
3      if(f[x]!=x)f[x]=find(f[x]);
4      return f[x];
5  }
6  void merge_set(int x,int y)
7  {
8      int fx=find(x),fy=find(y);
9      if(fx!=fy)
10     {
11         siz[fy]+=siz[fx];//维护大小
12         f[fx]=fy;
13     }
14 }
15 void init()
16 {
17     for(int i=1;i<=n;i++)
18     {
19         f[i]=i;
20         siz[i]=1;
21     }
22 }

```

维护距离(向量本质, 有向图)

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=1e6+10;
6  int n,k,cnt;
7  int f[N],d[N];
8  int find(int x)
9  {
10     if(f[x]!=x)
11     {
12         int t=find(f[x]);
13         d[x]+=d[f[x]];
14         f[x]=t;
15     }
16     return f[x];
17 }
18 int main()
19 {
20     cin>>n>>k;
21     for(int i=1;i<=n;i++)f[i]=i;
22     while(k-->0)
23     {
24         int op,x,y;
25         cin>>op>>x>>y;
26         if(x>n||y>n)cnt++;
27         else
28         {
29             int fx=find(x),fy=find(y);
30             if(op==1)//同类
31             {
32                 if(fx==fy&&(d[x]-d[y])%3)cnt++;
33                 else if(fx!=fy)
34                 {
35                     f[fx]=fy;
36                     d[fx]=d[y]-d[x];
37                 }
38             }
39             else
40             {
41                 if(fx==fy&&(d[x]-d[y]-1)%3)cnt++;
42                 else if(fx!=fy)
43                 {
44                     f[fx]=fy;
45                     d[fx]=d[y]+1-d[x];
46                 }
47             }
48         }
49     }
50     cout<<cnt;
51     return 0;
52 }
```


维护大小

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=1e6+10;
4
5
6  int n,m;
7  int f[N],siz[N];
8  int find(int x)
9  {
10     if(f[x]!=x)f[x]=find(f[x]);
11     return f[x];
12 }
13 int main()
14 {
15     scanf("%d%d",&n,&m);
16     for(int i=1;i<=n;i++)//看清题目可能从0开始
17     {
18         f[i]=i;
19         siz[i]=1;
20     }
21     while(m-->0)
22     {
23         int a,b;
24         char op[2];
25         scanf("%s",op);
26         if(op[0]=='c')
27         {
28             scanf("%d%d",&a,&b);
29             if(find(a)==find(b))continue;//判断一下有没有在同一集合里了
30             siz[find(b)]+=siz[find(a)];//要在合并之前
31             f[find(a)]=find(b);
32         }
33         else if(op[1]=='1')
34         {
35             scanf("%d",&a);
36             if(find(a)==find(b))puts("Yes");
37             else puts("No");
38         }
39         else
40         {
41             scanf("%d",&a);
42             printf("%d\n",siz[find(a)]);
43         }
44     }
45     return 0;
46 }
```

扩展域

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4
```

```

5  const int N=1e6+10;
6  int f[N],enem[N]; //enem存p的敌人
7  int find(int x)
8  {
9      if(f[x]!=x)f[x]=find(f[x]);
10     return f[x];
11 }
12 void merge_set(int x,int y)
13 {
14     int fx=find(x),fy=find(y);
15     if(fx==fy)return; //不要忘记
16     else f[fx]=fy;
17 }
18 int main()
19 {
20     int n,m,cnt;
21     char op[2];
22     scanf("%d%d",&n,&m);
23     cnt=0;
24     for(int i=1;i<=2*n;i++)f[i]=i;
25     for(int i=1;i<=m;i++)
26     {
27         int p,q;
28         scanf("%s%d%d",op,&p,&q);
29         int fp=find(p),fq=find(q);
30         if(op[0]=='F')merge_set(p,q);
31         else
32         {
33             if(!enem[p])enem[p]=q;
34             else merge_set(q,enem[p]);
35             if(!enem[q])enem[q]=p;
36             else merge_set(p,enem[q]);
37         }
38     }
39     for(int i=1;i<=n;i++)
40     {
41         if(f[i]==i)cnt++;
42     }
43     printf("%d",cnt);
44     return 0;
45 }

```

链表

普通链表

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=1e6+5;
6  int head,e[N],ne[N],idx; //把e和ne想成节点idx的两个属性
7  //初始化
8  void init()
9  {

```

```

10     head=-1;
11     idx=0;//0开头
12 }
13 //链表头插入x
14 void add_to_head(int x)
15 {
16     e[idx]=x,ne[idx]=head,head=idx++;//最后一个重新把head指向了idx，因为开局head
    为-1嘛
17 }
18 //在k后面插入一个点
19 void add(int k,int x)
20 {
21     e[idx]=x,ne[idx]=ne[k],ne[k]=idx++;//先用idx在+1
22 }
23 //移除k后面的点
24 void remove(int k)
25 {
26     ne[k]=ne[ne[k]];
27 }
28 //移除头节点，要保证头节点存在
29 void remove_head()
30 {
31     head=ne[head];
32 }
33
34
35 int main()
36 {
37     int m,k,x;
38     cin>>m;
39     char op;
40     init();//不要忘记初始化
41     while(m--)
42     {
43         cin>>op;
44         if(op=='H')
45         {
46             cin>>x;
47             add_to_head(x);
48         }
49         else if(op=='D')
50         {
51             cin>>k;
52             if(!k)remove_head();
53             else remove(k-1);    //因为0开头的，所以第k个下标是k-1
54         }
55         else if(op=='I')
56         {
57             cin>>k>>x;
58             add(k-1,x);
59         }
60     }
61     for(int i=head;i!=-1;i=ne[i])cout<<e[i]<<" ";//链表输出方式,记住是ne[i]和
    i!=-1
62
63
64     return 0;
65

```

```
66 |
67 | }
```

双链表

哈希表

堆

查找

二分查找

二分的应用

1. 数据范围大，数据量小，把数据存到普通数组，排序，二分查找下标，可以得到区间数据数量

整数二分

```
1  bool check1(ll mid){}
2
3  ll bs1(ll l,ll r)//第一个满足条件的
4  {
5      while(l<r)
6      {
7          ll mid=l+r>>1;
8          if(check1(mid))r=mid;
9          else l=mid+1;
10     }
11     return l;
12 }
13 bool check2(ll mid){}
14 ll bs2(ll l,ll r)//最后一个满足条件的
15 {
16     while(l<r)
17     {
18         ll mid=l+r+1>>1;
19         if(check2(mid))l=mid;
20         else r=mid-1;
21     }
22     return l;
23 }
```

浮点数二分

```
1  bool check(double x) { /* ... */ } // 检查x是否满足某种性质
2
3
4  double bsearch_3(double l, double r)//输入l和r的时候保证l<r不要输入一个负数就反过来了不能写(-n,n)
5  {
6      const double eps = 1e-6;    // eps 表示精度，取决于题目对精度的要求，一般比要求的两位有效数字
7      while (r - l > eps)
```

```

8     {
9         double mid = (l + r) / 2;
10        if (check(mid)) r = mid;
11        else l = mid;
12    }
13    return l;
14 }

```

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const double eps=1e-7;
4  double y;
5  double f(double x)
6  {
7      return 0.0001*x*x*x*x*x+0.003*x*x*x+0.5*x-3;
8  }
9  int main()
10 {
11     while(scanf("%lf",&y)!=EOF)
12     {
13         double mid;
14         double l=-20.0,r=20.0;
15         while(l<=r)
16         {
17             mid=(l+r)/2.0;
18             if(fabs(f(mid)-y)<1e-5)break;//如果直接数值型的可以这样处理保证精度
19             if(f(mid)<y)l=mid;
20             else r=mid;
21         }
22         printf("%.4lf\n",mid);
23     }
24     return 0;
25 }

```

搜索与图论

图

图的存储

邻接表

```

1 // 对于每个点k，开一个单链表，存储k所有可以走到的点。h[k]存储这个单链表的头结点
2 int h[N], e[N], ne[N], idx;
3
4 // 添加一条边a->b
5 void add(int a, int b)
6 {
7     e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
8 }
9
10 // 初始化
11 idx = 0;
12 memset(h, -1, sizeof h);

```

邻接矩阵

建图小技巧

1. 反向建图日神仙
- 2.

DFS

有多少个连通块（洪泛）

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int n,m,cnt;
5 char mp[505][505];
6 int xx[]={0,0,1,-1};
7 int yy[]={1,-1,0,0};
8
9 void dfs(int x,int y)
10 {
11     for(int i=0;i<4;i++)
12     {
13         int dx=x+xx[i];
14         int dy=y+yy[i];
15         if(dx>=0&&dx<=n+1&&dy>=0&&dy<=m+1&&mp[dx][dy]!='*')
16         {
17             mp[dx][dy]='*';//直接标记
18             dfs(dx,dy);
19         }
20     }
21 }
22
23 int main()
24 {
25     cin>>n>>m;
26     for(int i=1;i<=n;i++)
27     {
28         for(int j=1;j<=m;j++)
29         {
30             char c;
31             cin>>c;
32             mp[i][j]=c;
33         }
34     }
35 }

```

```

34     }
35     dfs(0,0); //方式开局就是这*
36     for(int i=1;i<=n;i++)
37         for(int j=1;j<=m;j++)
38             if(mp[i][j]=='0')
39                 cnt++;
40
41     cout<<cnt;
42     return 0;
43 }

```

DFS遍历图(树的重心)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=1e6+10;
6  const int M=N*2; //无向图两条边
7
8
9  int h[N],e[M],ne[M],idx;
10 bool st[N];
11 int ans=N;
12 int n;
13 void add(int a,int b) //a指向b
14 {
15     e[idx]=b,ne[idx]=h[a],h[a]=idx++;
16 }
17 //以u为根的子树大小（点数量）
18 int dfs(int u)
19 {
20     st[u]=1; //标记一下
21     int sum=1,res=0; //sum
22     for(int i=h[u];i!=-1;i=ne[i]) //遍历与u连通的点
23     {
24         int j=e[i];
25         if(!st[j])
26         {
27             int s=dfs(j); //当前子树大小
28             res=max(res,s);
29             sum+=s; //s是u为根子数大小一部分
30         }
31     }
32     res=max(res,n-sum); //n-sum为，子树上面一坨
33     ans=min(ans,res);
34     return sum; //以u为根子节点大小
35 }
36 int main()
37 {
38     cin>>n;
39     memset(h,-1,sizeof(h)); //初始化
40     for(int i=1;i<n;i++)
41     {
42         int a,b;
43         cin>>a>>b;
44         add(a,b),add(b,a); //无向图两条边

```

```

45     }
46     dfs(1); //图当中的编号开始搜，随便从哪个点开始都可以
47     cout<<ans<<endl;
48     return 0;
49 }

```

列出所有解

全排列

```

1
2
3 #include<bits/stdc++.h>
4 using namespace std;
5
6
7 int n;
8 bool vis[30];
9 int a[20];
10
11
12 void pr()
13 {
14     for(int i=1;i<=n;i++)
15     {
16         cout<<setw(5)<<a[i];
17     }
18     cout<<endl;
19 }
20 void dfs(int x) //x是层数
21 {
22     if(x>n)
23     {
24         pr(); //超出了n就结束了
25     }
26     for(int i=1;i<=n;i++)
27     {
28         if(!vis[i])
29         {
30             a[x]=i; //第x层是i;
31             vis[i]=1;
32             dfs(x+1);
33             vis[i]=0; //释放回到上一个节点，消去访问记录，其实a[x]也要消去只不过新的值
会覆盖
34         }
35     }
36 }
37
38
39 int main()
40 {
41     cin>>n;
42     dfs(1);
43     return 0;
44 }
45

```


STL

```
1 //prev_permutation函数可以制造前一个排列，如果已经为第一个，则返回false。
2 #include<bits/stdc++.h>
3 using namespace std;
4 int n,a[10000];
5 int main()
6 {
7     cin>>n;
8     for(int i=0;i<n;i++) //读入数据
9         cin>>a[i];
10    if(prev_permutation(a,a+n)) //如果为真就输出数组
11        for(int i=0;i<n;i++)
12            cout<<a[i]<<" ";
13    else cout<<"ERROR"; //否则输出ERROR
14    cout<<endl;
15    return 0;
16 }
17 //next_permutation同理
18 int main()
19 {
20     string str = "abcde";
21     int num = 1;
22     while(next_permutation(str.begin(),str.end()))
23     {
24         num++;
25         cout<<str<<endl;
26         if(num==5)
27             break;
28     }
29     return 0;
30 }
```

组合输出

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4
5 int n,r;
6 int a[50];
7 bool vis[50];
8
9
10 void pr()
11 {
12     for(int i=1;i<=r;i++)
13         cout<<setw(3)<<a[i];
14     cout<<endl;
15 }
16
17
18 void dfs(int x)
19 {
20     if(x>r)
21     {
22         pr();
```

```

23         return;
24     }
25     for(int i=1;i<=n;i++)
26     {
27         if(!vis[i]&&(i>a[x-1]||x==1))
28         {
29             a[x]=i;
30             vis[i]=1;
31             dfs(x+1);
32             vis[i]=0;
33         }
34     }
35 }
36 int main()
37 {
38     cin>>n>>r;
39     dfs(1);
40     return 0;
41 }

```

BFS

最短步数（边的权值均为1，STL写法）

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  int l,r,c;
4  int xx[]={1,-1,0,0,0,0};
5  int yy[]={0,0,1,-1,0,0};
6  int zz[]={0,0,0,0,1,-1};
7  int sx,sy,sz,ex,ey,ez;
8  char mp[40][40][40];
9  bool vis[40][40][40];
10 bool flag;
11 struct node
12 {
13     int x,y,z,s;//s存步数
14 };
15
16
17 void bfs(int z,int x,int y)
18 {
19     queue<node>q;
20     q.push((node){x,y,z,0}); //创建结构体队列
21     vis[z][x][y]=1; //不要忘记
22     while(!q.empty())
23     {
24         if(q.front().x==ex&&q.front().y==ey&&q.front().z==ez)
25         {
26             flag=1;
27             printf("Escaped in %d minute(s).",q.front().s);
28             break;
29         }
30         for(int i=0;i<6;i++)
31         {
32             int dx=q.front().x+xx[i]; //是队头的xyz不是x+xx[i]
33             int dy=q.front().y+yy[i];

```

```

34         int dz=q.front().z+zz[i];
35         //看清地图范围0~n-1还是1~n, 看清是n*n还是n*m, 哪个是行哪个是列也要看清楚
36         if(dx>=1&&dy>=1&&dz>=1&&dz<=1&&dx<=r&&dy<=c&&!vis[dz][dx]
[dy]&&mp[dz][dx][dy]!='#')
37         {
38             q.push((node){dx,dy,dz,q.front().s+1});
39             vis[dz][dx][dy]=1;
40         }
41     }
42     q.pop();
43 }
44 }
45 int main()
46 {
47     cin>>l>>r>>c;
48     for(int i=1;i<=l;i++)
49     {
50         for(int j=1;j<=r;j++)
51         {
52             for(int k=1;k<=c;k++)
53             {
54                 cin>>mp[i][j][k];
55                 if(mp[i][j][k]=='S')
56                 {
57                     sz=i;sx=j;sy=k;
58                 }
59                 if(mp[i][j][k]=='E')
60                 {
61                     ez=i;ex=j;ey=k;
62                 }
63             }
64         }
65     }
66     bfs(sz,sx,sy);
67     if(!flag)printf("Trapped!");
68     return 0;
69 }

```

BFS+路径保存

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  struct node
5  {
6      int x,y,s;
7  };
8
9  int n,m;
10 int xx[]={0,0,1,-1};
11 int yy[]={1,-1,0,0};
12 const int N=105;
13 node pre[N][N]; //保存路径
14 int g[N][N]; //保存图
15 bool vis[N][N];
16 void bfs(int sx,int sy)
17 {

```

```

18     queue<node>q;
19     q.push((node){1,1,0});
20     vis[sx][sy]=1; //不要忘记
21     pre[sx][sy]=(node){1,1,0};
22     while(!q.empty())
23     {
24         if(q.front().x==n&&q.front().y==m)
25         {
26             cout<<q.front().s<<endl;
27             cout<<n<<" "<<m<<endl;
28             while(n!=sx||m!=sy)//输出路径
29             {
30                 cout<<pre[n][m].x<<" "<<pre[n][m].y<<endl;
31                 n=pre[n][m].x;
32                 m=pre[n][m].y;
33             }
34             break;
35         }
36         for(int i=0;i<4;i++)
37         {
38             int dx=q.front().x+xx[i];
39             int dy=q.front().y+yy[i];
40             if(dx>=1&&dy>=1&&dx<=n&&dy<=m&&!g[dx][dy]&&!vis[dx][dy])
41             {
42                 pre[dx][dy]=(node){q.front().x,q.front().y,q.front().s}; //保
留从哪里转移过来的就行
43                 q.push((node){dx,dy,q.front().s+1});
44                 vis[dx][dy]=1;
45             }
46         }
47         q.pop();
48     }
49 }
50
51 int main()
52 {
53     cin>>n>>m;
54     for(int i=1;i<=n;i++)
55         for(int j=1;j<=m;j++)
56             cin>>g[i][j];
57     bfs(1,1);
58 }

```

BFS遍历图(边权1最短路, 手动模拟队列写法)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=1e6+10;
6  int h[N],e[N],ne[N],idx;
7  int n,m;
8  int d[N],q[N]; //d距离, q队列
9
10
11 void add(int a,int b)
12 {

```

```

13     e[idx]=b,ne[idx]=h[a],h[a]=idx++;
14 }
15
16
17 int bfs()
18 {
19     int hh=0,tt=0;
20     q[0]=1;//0号节点是编号为1的点，q[0]=v可以做v开始搜的广搜
21     memset(d,-1,sizeof(d));
22     d[1]=0;//存储每个节点离起点的距离，这个不要忘记了
23     while(hh<=tt)
24     {
25         int t=q[hh++];//t=q[hh]队头同时hh+1弹出队头
26         for(int i=h[t];i!=-1;i=ne[i])
27         {
28             int j=e[i];
29             //如果j没被扩展过
30             if(d[j]==-1)
31             {
32                 d[j]=d[t]+1;//d[j]存储j离起点距离，并标记访问过
33                 q[++tt]=j;//压入j
34             }
35         }
36     }
37     return d[n];
38 }
39 int main()
40 {
41     cin>>n>>m;
42     memset(h,-1,sizeof(h));
43     for(int i=1;i<=m;i++)
44     {
45         int a,b;
46         cin>>a>>b;
47         add(a,b);
48     }
49     cout<<bfs()<<endl;
50     return 0;
51 }

```

连通块里多少块

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  int xx[]={0,0,1,-1};
4  int yy[]={1,-1,0,0};
5  int vis[1005][1005];
6  bool mp[1005][1005];
7  int ans[1000005];
8  int color,cnt,n,m;
9
10
11 void bfs(int x, int y)
12 {
13     queue<int>h;
14     queue<int>l;
15     h.push(x);l.push(y);

```

```

16     vis[x][y]=color;
17     while(!h.empty())
18     {
19         for(int i=0;i<4;i++)
20         {
21             int dx=h.front()+xx[i];
22             int dy=l.front()+yy[i];
23             if(dx>=1&&dx<=n&&dy>=1&&dy<=n&&mp[dx][dy] !=mp[h.front()]
[1.front()]&&!vis[dx][dy])
24             {
25                 h.push(dx);l.push(dy);
26                 vis[dx][dy]=color;//颜色标记区分不同的连通块
27             }
28         }
29         h.pop();l.pop();//弹出多少次就有多少格子
30         cnt++;
31     }
32     return;
33 }
34 int main()
35 {
36     cin>>n>>m;
37     for(int i=1;i<=n;i++)
38     {
39         for(int j=1;j<=n;j++)
40         {
41             char ch;
42             cin>>ch;
43             if(ch=='1')mp[i][j]=1;
44             else mp[i][j]=0;
45         }
46     }
47     //如果是确定的起点那下面的直接bfs(sx,sy)即可
48     for(int i=1;i<=n;i++)
49     {
50         for(int j=1;j<=n;j++)
51         {
52             if(!vis[i][j])//排除已经搜索过的连通块
53             {
54                 color++;
55                 bfs(i,j);
56                 ans[color]=cnt;
57                 cnt=0;//初始化cnt
58             }
59         }
60     }
61     for(int i=1;i<=m;i++)
62     {
63         int x,y;
64         cin>>x>>y;
65         cout<<ans[vis[x][y]]<<endl;
66     }
67     return 0;
68 }

```

拓扑序列 (有向无环图AOV)

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=1e6+10;
4  int n,m;
5  int h[N],e[N],ne[N],idx;
6  int q[N],d[N]; //q队列存储层次遍历序列，d存储i号节点入度
7
8
9  void add(int a,int b)
10 {
11     e[idx]=b,ne[idx]=h[a],h[a]=idx++;
12 }
13 //返回布尔序列是否存在，若存在，则存储在q数组中
14 bool topsort()
15 {
16     int hh=0,tt=-1;
17     //遍历每个节点，入队为0则入队
18     for(int i=1;i<=n;i++)
19         if(!d[i])
20             q[++tt]=i;
21
22     while(hh<=tt)
23     {
24         //队列不为空则取出头节点
25         int t=q[hh++]; //出队的顺序就是拓扑序
26         //遍历头节点每个出边
27         for(int i=h[t];i!=-1;i=ne[i])
28         {
29             int j=e[i];
30             //出边能到的节点入度减1
31             d[j]--;
32             if(d[j]==0)q[++tt]=j; //如果节点j，入度0则入队
33         }
34     }
35
36     return tt==n-1; //不要打成=，所有点都入队了说明存在拓扑序列
37 }
38 int main()
39 {
40     cin>>n>>m;
41     memset(h,-1,sizeof(h));
42     for(int i=0;i<m;i++)
43     {
44         int a,b;
45         cin>>a>>b;
46         add(a,b);
47         d[b]++; //b节点入度增加1
48     }
49     if(topsort())
50     {
51         for(int i=0;i<n;i++)printf("%d ",q[i]);
52         puts("");
53     }
54     else puts("-1");
55     return 0;
```

最短路

dijkstra朴素稠密图 $O(n^2)$

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=510;
6  int n,m;
7  int g[N][N]; //邻接矩阵处理稠密图
8  int dist[N];
9  bool st[N];
10 int dijkstra()
11 {
12     memset(dist,0x3f,sizeof(dist)); //距离初始化为正无穷
13     memset(st,0,sizeof st);
14     dist[1]=0; //一号点初始化为0
15
16     for(int i=0;i<n;i++) //迭代n次
17     {
18         int t=-1; //t开始为-1表示还没确定最短路
19         for(int j=1;j<=n;j++)
20             if(!st[j]&&(t==-1||dist[t]>dist[j])) //所有st[j]=0的点中找到距离最小
21                 t=j;
22         st[t]=1;
23
24         for(int j=1;j<=n;j++) //用t更新其他点到1的距离，遍历边有效更新m次
25             dist[j]=min(dist[j],dist[t]+g[t][j]);
26     }
27
28     if(dist[n]==0x3f3f3f3f) return -1;
29     return dist[n];
30 }
31 int main()
32 {
33     scanf("%d%d",&n,&m);
34
35     memset(g,0x3f,sizeof(g)); //初始化点位无穷
36
37     while(m--)
38     {
39         int a,b,c;
40         scanf("%d%d%d",&a,&b,&c);
41         g[a][b]=min(g[a][b],c); //处理重边保留距离最短的即可
42     }
43
44     int t=dijkstra();
45
46     printf("%d\n",t);
47
48     return 0;
49 }

```


dijkstra堆优化稀疏图 $O(m\log n)$

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef pair<int,int> PII;
4  int n,m,s;
5  const int N=5e5+10;//邻接表N看边数啊啊啊
6  int h[N],e[N],ne[N],w[N],idx;
7  int dist[N];
8  bool vis[N];
9  void add(int a,int b,int c)
10 {
11     e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
12 }
13 void dijkstra(int st)
14 {
15     memset(dist,0x3f,sizeof dist);//你开0x3f就要对于int的数组，不要用long long数
    组开memset 0x3f
16     memset(vis,0,sizeof vis);
17     dist[st]=0;
18     priority_queue<PII,vector<PII>,greater<PII>>heap;//小根堆，顺序不能换，因为
    pair按first排序
19     heap.push({0,st});//first距离，second编号
20     while(heap.size())
21     {
22         //维护当前未被st标记且离源点最近的点
23         auto t=heap.top();
24         heap.pop();
25         int ver=t.second,distance=t.first;
26         if(vis[ver])continue;
27         vis[ver]=1;
28         for(int i=h[ver];i!=-1;i=ne[i])//用t更新其他点
29         {
30             int j=e[i];
31             if(dist[j]>distance+w[i])
32             {
33                 dist[j]=distance+w[i];//松弛
34                 heap.push({dist[j],j});
35             }
36         }
37     }
38 }
39 int main()
40 {
41     ios::sync_with_stdio(0);
42     cin.tie(0);
43     cin>>n>>m>>s;
44     memset(h,-1,sizeof h);
45     while(m-->0)
46     {
47         int a,b,c;
48         cin>>a>>b>>c;
49         add(a,b,c);
50     }
51     dijkstra(s);
```

```

52     for(int i=1;i<=n;i++)
53     {
54         if(dist[i]!=0x3f3f3f3f)cout<<dist[i]<<" ";
55         else cout<<"2147483647 ";
56     }
57     return 0;
58 }

```

dijkstra反向建图求多个点到起点的最短路

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=1e4+10,M=2e5+10;
5  int h[N],e[M],w[M],ne[M],idx;
6  int n,m;
7  int dist[N];
8  bool vis[N];
9  typedef pair<int,int> PII;
10 void add(int a,int b,int c)
11 {
12     e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
13 }
14 void dijkstra(int st)
15 {
16     memset(dist,0x3f,sizeof dist);
17     memset(vis,0,sizeof vis);
18     dist[st]=0;
19     priority_queue<PII,vector<PII>,greater<PII>>heap;
20     heap.push({0,st});
21     while(heap.size())
22     {
23         auto t=heap.top();
24         int ver=t.second,distance=t.first;
25         heap.pop();
26         if(vis[ver])continue;
27         vis[ver]=1;
28         for(int i=h[ver];i!=-1;i=ne[i])
29         {
30             int j=e[i];
31             if(dist[j]>distance+w[i])
32             {
33                 dist[j]=distance+w[i];
34                 heap.push({dist[j],j});
35             }
36         }
37     }
38 }
39
40 int main()
41 {
42     ios::sync_with_stdio(0);
43     cin.tie(0);
44     cout.tie(0);
45     cin>>n>>m;
46     memset(h,-1,sizeof h);
47     while(m--)

```

```

48     {
49         int a,b,c;
50         cin>>a>>b>>c;
51         add(a,b,c);
52         add(b+n,a+n,c);

53     }
54     dijkstra(1);
55     int res=0;
56     for(int i=2;i<=n;i++)res+=dist[i];
57     dijkstra(n+1);
58     for(int i=n+2;i<=2*n;i++)res+=dist[i];
59     cout<<res;
60 }

```

bellman_ford()(处理边数限制)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=510,M=10010;
6
7
8  int n,m,k;
9  int dist[N],backup[N];
10
11
12 struct Edge
13 {
14     int a,b,w;
15 }edges[M];
16
17
18 int bellman_ford()
19 {
20     memset(dist,0x3f,sizeof dist);
21     dist[1]=0;//初始化
22     for(int i=0;i<k;i++)
23     {
24         memcpy(backup,dist,sizeof dist);//防止串联更新
25         for(int j=0;j<m;j++)
26         {
27             int a=edges[j].a,b=edges[j].b,w=edges[j].w;
28             dist[b]=min(dist[b],backup[a]+w);//用备份更新
29         }
30     }
31     if(dist[n]>0x3f3f3f3f/2)return -1;//
32     return dist[n];
33 }
34 int main()
35 {
36     ios::sync_with_stdio(0);
37     cin.tie(0);
38     cin>>n>>m>>k;

```

```

39
40     for(int i=0;i<m;i++)
41     {
42         int a,b,w;
43         cin>>a>>b>>w;
44         edges[i]={a,b,w};
45     }
46     int t=bellman_ford();
47     if(t== -1)puts("impossible");
48     else cout<<t<<endl;
49     return 0;
50 }

```

SPFA

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=1e5+10;
4  int h[N],e[N],w[N],ne[N],idx;
5  bool st[N];
6  int dist[N];
7  int n,m;
8  void add(int a,int b,int c)
9  {
10     e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
11 }
12
13
14 int spfa()
15 {
16     memset(dist,0x3f,sizeof dist);
17     dist[1]=0;
18     queue<int>q;
19     q.push(1);
20     st[1]=1;
21     while(q.size())
22     {
23         int t=q.front();
24         q.pop();
25         st[t]=0;
26         for(int i=h[t];i!=-1;i=ne[i])
27         {
28             int j=e[i];
29             if(dist[j]>dist[t]+w[i])
30             {
31                 dist[j]=dist[t]+w[i];
32                 if(!st[j])
33                 {
34                     q.push(j);
35                     st[j]=1;
36                 }
37             }
38         }
39     }
40     if(dist[n]>0x3f3f3f3f/2)return -1;
41     return dist[n];
42 }

```

```

43 int main()
44 {
45     ios::sync_with_stdio(0);
46     cin.tie(0);
47     cin>>n>>m;
48     memset(h,-1,sizeof h);
49     while(m--)
50     {
51         int a,b,c;
52         cin>>a>>b>>c;
53         add(a,b,c);
54     }
55     int t=spfa();
56     if(t==-1)puts("impossible");
57     else cout<<t<<endl;
58     return 0;
59 }

```

SPFA (判有无负权环)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=2010,M=10010;
6  int h[N],e[M],ne[M],w[M],idx;
7  int dist[N],cnt[N];
8  bool st[N];
9  int n,m;
10 void add(int a,int b,int c)
11 {
12     e[idx]=b,w[idx]=c,ne[idx]=h[a],h[a]=idx++;
13 }
14 bool spfa()
15 {
16     queue<int>q;
17     for(int i=1;i<=n;i++)q.push(i),st[i]=1;
18     while(q.size())
19     {
20         int t=q.front();
21         q.pop();
22         st[t]=0;
23         for(int i=h[t];i!=-1;i=ne[i])
24         {
25             int j=e[i];
26             if(dist[j]>dist[t]+w[i])
27             {
28                 dist[j]=dist[t]+w[i];
29                 cnt[j]=cnt[t]+1;//不要写++cnt[j], 重边会影响的
30                 if(cnt[j]>=n)return 1;
31                 if(!st[j])
32                 {
33                     q.push(j);
34                     st[j]=1;
35                 }
36             }
37         }
38     }
39 }

```

```

38     }
39     return 0;
40 }
41 int main()
42 {
43     ios::sync_with_stdio(0);
44     cin.tie(0);
45     cin>>n>>m;
46     memset(h,-1,sizeof h);
47     while(m--)
48     {
49         int a,b,c;
50         cin>>a>>b>>c;
51         add(a,b,c);
52     }
53     if(spfa())cout<<"Yes"<<endl;
54     else cout<<"No"<<endl;
55     return 0;
56 }

```

Floyd(多源汇最短路)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  int n,m,q;
4  const int N=210,INF=1e9;
5  int d[N][N];
6
7  void floyd()
8  {
9      for(int k=1;k<=n;k++)
10         for(int i=1;i<=n;i++)
11             for(int j=1;j<=n;j++)
12                 d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
13 }
14 int main()
15 {
16     ios::sync_with_stdio(0);
17     cin.tie(0);
18     cin>>n>>m>>q;
19     for(int i=1;i<=n;i++)
20         for(int j=1;j<=n;j++)
21             if(i==j)d[i][j]=0;//处理自环
22             else d[i][j]=INF;
23
24     while(m--)
25     {
26         int a,b,w;
27         cin>>a>>b>>w;
28         d[a][b]=min(d[a][b],w);
29     }
30     floyd();
31     while(q--)
32     {
33         int a,b;
34         cin>>a>>b;
35         if(d[a][b]>INF/2)cout<<"impossible"<<endl;

```

```

36         else cout<<d[a][b]<<endl;
37     }
38     return 0;
39 }

```

Floyd求最短环

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=200;
4  typedef long long ll;
5  const int INF=0x7f7f7f7f;
6  ll g[N][N],dist[N][N];
7  int n,m;
8  int main()
9  {
10     ios::sync_with_stdio(0);
11     cin.tie(0);
12     cout.tie(0);
13     cin>>n>>m;
14     for(int i=1;i<=n;i++)
15         for(int j=1;j<=n;j++)
16             if(i!=j)dist[i][j]=g[i][j]=INF;
17     while(m--)
18     {
19         ll a,b,c;
20         cin>>a>>b>>c;
21         g[a][b]=g[b][a]=min(g[a][b],c);
22         dist[a][b]=dist[b][a]=min(g[a][b],c);
23     }
24     ll ans=INF;
25     for(int k=1;k<=n;k++)
26     {
27         for(int i=1;i<k;i++)
28             for(int j=i+1;j<k;j++)
29                 ans=min(ans,dist[i][j]+g[i][k]+g[k][j]);
30         for(int i=1;i<=n;i++)
31             for(int j=1;j<=n;j++)
32             {
33                 dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
34                 dist[j][i]=dist[i][j];
35             }
36     }
37     if(ans==INF)cout<<"No solution.";
38     else cout<<ans;
39     return 0;
40 }

```

最小生成树

Kruskal (稀疏图) (O(mlogm))

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=2e6+10;
4  int n,m;
5  int f[N];

```

```

6 struct Edge
7 {
8     int a,b,w;
9 }edges[N];
10 bool cmp(Edge a,Edge b)
11 {
12     return a.w<b.w;//如果写成if的最后别忘记加return 0
13 }
14
15 int find(int x)
16 {
17     if(f[x]!=x)f[x]=find(f[x]);
18     return f[x];
19 }
20
21 int main()
22 {
23     ios::sync_with_stdio(0);
24     cin.tie(0);
25     cin>>n>>m;
26     for(int i=0;i<m;i++)
27     {
28         int a,b,c;
29         cin>>a>>b>>c;
30         edges[i]={a,b,c};
31     }
32     sort(edges,edges+m,cmp);//对边排序
33     int res=0,cnt=0;//res最小生成树边权重之和, cnt记录最小生成树中的边数
34     //并查集看有没有在集合里
35     for(int i=0;i<=m;i++)f[i]=i;
36     for(int i=0;i<m;i++)
37     {
38         int a=edges[i].a,b=edges[i].b,w=edges[i].w;
39         a=find(a),b=find(b);
40         if(a!=b)
41         {
42             f[a]=b;
43             res+=w;
44             cnt++;
45         }
46     }
47     if(cnt<n-1)cout<<"impossible"<<endl;
48     else cout<<res<<endl;
49     return 0;
50 }

```

Prim(稠密图)($O(n^2)$)

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int N=1e3,INF=0x3f3f3f3f;
4 int dist[N],g[N][N];//dist是顶点到任意一个树顶点的最短距离
5 bool st[N];
6 int n,m;
7 int prim()
8 {
9     memset(dist,0x3f,sizeof dist);

```



```

10     int res=0;//存最小生成树所有边长度之和
11     for(int i=0;i<n;i++)
12     {
13         int t=-1;
14         for(int j=1;j<=n;j++)//找集合外所有点中到集合距离最小的点
15             if(!st[j]&&(t==-1||dist[t]>dist[j]))
16                 t=j;
17         if(i&&dist[t]==INF)return INF;//不是第一个点而且最短距离都为INF，就不存在最
最小生成树
18         if(i)res+=dist[t];//只要不是第一个点
19         st[t]=1;
20         //扫描顶点t的所有边，在以t为中心更新其他点到树的距离（这时候t已经在生成树里了，其
他点到t距离就是到生成树距离）
21         for(int j=1;j<=n;j++)dist[j]=min(dist[j],g[t][j]);
22     }
23     return res;
24 }
25 int main()
26 {
27     ios::sync_with_stdio(0);
28     cin.tie(0);
29     cin>>n>>m;
30     memset(g,0x3f,sizeof g);
31     while(m--)
32     {
33         int a,b,c;
34         cin>>a>>b>>c;
35         g[a][b]=g[b][a]=min(g[a][b],c);
36     }
37     int t=prim();
38     if(t==INF)cout<<"impossible";
39     else cout<<t<<endl;
40     return 0;
41 }

```

二分图

染色法判二分图($O(m+n)$)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=1e6+10,M=2e6+10;
5
6  int n,m;
7  int h[N],e[M],ne[M],idx;
8  int color[N];
9
10 bool dfs(int u,int c)//u为点编号，c为染色
11 {
12     color[u]=c;
13     for(int i=h[u];i!=-1;i=ne[i])//遍历和点连接的点
14     {
15         int j=e[i];
16         if(!color[j])//没染色，那就染（3-c实现1染2，2染1）
17         {
18             if(!dfs(j,3-c))return 0;

```

```

19     }
20     else if(color[j]==c)return 0;//已经染色
21 }
22 return 1;
23 }
24 void add(int a,int b)
25 {
26     e[idx]=b,ne[idx]=h[a],h[a]=idx++;
27 }
28 int main()
29 {
30     ios::sync_with_stdio(0);
31     cin.tie(0);
32     cin>>n>>m;
33     memset(h,-1,sizeof h);
34     while(m--)
35     {
36         int a,b;
37         cin>>a>>b;
38         add(a,b);
39         add(b,a);
40     }
41     bool flag=1;//染色是否有矛盾发生
42     for( int i=1;i<=n;i++)
43         if(!color[i])
44         {
45             if(!dfs(i,1))//dfs false有矛盾发生
46             {
47                 flag=0;
48                 break;
49             }
50         }
51
52     if(flag)puts("Yes");
53     else puts("No");
54 }

```

匈牙利算法二分图最大匹配图

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=510,M=100010;
5
6  int n1,n2,m;
7  int h[N],e[M],ne[M],idx;
8  int match[N];
9  bool st[N];
10
11 void add(int a,int b)
12 {
13     e[idx]=b,ne[idx]=h[a],h[a]=idx++;
14 }
15 bool find(int x)
16 {
17     for(int i=h[x];i!=-1;i=ne[i])//x是男的，j是妹子，遍历男的看上的所有妹子
18     {

```

```

19         int j=e[i];
20         if(!st[j])
21         {
22             st[j]=1;
23             if(match[j]==0||find(match[j]))//妹子没有匹配或者妹子原本匹配的男的有
备胎
24             {
25                 match[j]=x;
26                 return 1;
27             }
28         }
29     }
30     return 0;
31 }
32 int main()
33 {
34     ios::sync_with_stdio(0);
35     cin.tie(0);
36     cout.tie(0);
37     cin>>n1>>n2>>m;
38     memset(h,-1,sizeof h);
39     while(m--)
40     {
41         int a,b;
42         cin>>a>>b;
43         add(a,b);
44     }
45     int res=0;
46     for(int i=1;i<=n1;i++)
47     {
48         memset(st,0,sizeof st);
49         if(find(i))res++;
50     }
51     cout<<res<<endl;
52     return 0;
53 }

```

前缀和/差分

一维前缀和

```

1  S[i] = a[1] + a[2] + ... a[i]
2  a[1] + ... + a[r] = S[r] - S[1 - 1]

```

二维前缀和

```

1  S[i, j] = 第i行j列格子左上部分所有元素的和
2  以(x1, y1)为左上角, (x2, y2)为右下角的子矩阵的和为:
3  S[x2, y2] - S[x1 - 1, y2] - S[x2, y1 - 1] + S[x1 - 1, y1 - 1]

```

```

1  #include<bits/stdc++.h>
2  using namespace std;

```

```

3
4  const int N=1e3+5;
5  int mp[N][N],dp[N][N];
6  int main()
7  {
8      int n,m,q,x1,x2,y1,y2;
9      cin>>n>>m>>q;
10     for(int i=1;i<=n;i++)
11     {
12         for(int j=1;j<=m;j++)
13         {
14             cin>>mp[i][j];
15         }
16     }
17     memset(dp,0,sizeof(dp));
18
19     //预处理二位前缀和数组dp
20     for(int i=1;i<=n;i++)
21     {
22         for(int j=1;j<=m;j++)
23         {
24             dp[i][j]=dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1]+mp[i][j];
25         }
26     }
27
28
29     while(q-->0)
30     {
31         cin>>x1>>y1>>x2>>y2;
32         cout<<dp[x2][y2]-dp[x2][y1-1]-dp[x1-1][y2]+dp[x1-1][y1-1]<<endl;
33     }
34 }

```

Tip:前缀和和一些注意点(激光炸弹为例)

1. 卡内存，直接累加读入，开const in N的时候别太浪

```

1  cin>>s[i][j];
2  s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1];

```

2. 卡边界，为了方便处理前缀和，最好把前面的s[0]留出来，所以预处理

```

1  s[++x][++y]+=w;
2  //s[x++][y++]+=w; 错误

```

3. 覆盖范围处理(r不出界)

```

1  r=min(5001,r);
2  //5001为最大可能边界

```

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=5e3+10;

```

```

4  typedef long long ll;
5  int s[N][N];
6  int n,r,x,y,w;
7  int main()
8  {
9      cin>>n>>r;
10     r=min(5001,r); //预处理半径
11     while(n--)
12     {
13         cin>>x>>y>>w;
14         s[++x][++y]+=w;
15     }
16     for(int i=1;i<=5001;i++)
17         for(int j=1;j<=5001;j++)
18             s[i][j]+=s[i-1][j]+s[i][j-1]-s[i-1][j-1]; //直接累加节省内存
19
20     int res=0;
21     //枚举右下角
22     for(int i=r;i<=5001;i++) //直接开地图最大
23         for(int j=r;j<=5001;j++)
24             res=max(res,s[i][j]-s[i][j-r]-s[i-r][j]+s[i-r][j-r]);
25
26     cout<<res;
27     return 0;
28 }

```

一维差分

```

1  void init()
2  {
3      memset(b,0,sizeof b);
4      for(int i=1;i<=n;i++)b[i]=a[i]-a[i-1];
5  }
6  void edit(int l,int r,int c)
7  {
8      //区间预处理非常规情况
9      if(l>r)swap(l,r);
10     if(l<st)l=st;
11     if(r<st)r=st;
12     if(l>ed)l=ed;
13     if(r>ed)r=ed;
14     b[l]+=c;b[r+1]-=c;
15 }
16 void build()
17 {
18     for(int i=1;i<=n;i++)a[i]=a[i-1]+b[i];
19 }

```

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  //给区间[l, r]中的每个数加上c: B[l] += c, B[r + 1] -= c

```

```

5 //要确保修改的时候r+1>=l,l和r在区间范围内
6 const int N=1e6+5;
7 int a[N],b[N];
8 int main()
9 {
10     int n,l,r,c;
11     cin>>n;
12     for(int i=1;i<=n;i++)
13     {
14         cin>>a[i];
15         b[i]=a[i]-a[i-1];//构造差分数组
16     }
17     while(cin>>l>>r>>c)
18     {
19         b[l]+=c;
20         b[r+1]-=c;
21         for(int i=1;i<=n;i++)a[i]=a[i-1]+b[i];
22         for(int i=1;i<=n;i++)cout<<a[i]<<" ";//如果是连续差分求最终值,把这条放到
while外面就可以了,一般差分数据量比较大建议快速读入
23         cout<<endl;
24     }
25     return 0;
26 }

```

二维差分

1 给以(x1, y1)为左上角, (x2, y2)为右下角的子矩阵中的所有元素加上c:
2 $s[x1, y1] += c, s[x2 + 1, y1] -= c, s[x1, y2 + 1] -= c, s[x2 + 1, y2 + 1] += c$

```

1 #include<bits/stdc++.h>
2 using namespace std;
3
4
5 const int N=1e4+5;
6 int a[N][N],b[N][N];
7
8
9 inline int read()
10 {
11     int x=0,y=1;char c=getchar();//y代表正负(1.-1),最后乘上x就可以了。
12     while (c<'0' || c>'9') {if (c=='-') y=-1;c=getchar();}//如果c是负号就把y赋
为-1
13     while (c>='0'&&c<='9') x=x*10+c-'0',c=getchar();
14     return x*y;//乘起来输出
15 }
16
17
18 int main()
19 {
20     int n,m,q,x1,x2,y1,y2,c;
21     n=read(),m=read(),q=read();
22     for(int i=1;i<=n;i++)
23     {

```

```

24     for(int j=1;j<=m;j++)
25     {
26         a[i][j]=read();
27         b[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1]; //预处理差分
28     }
29 }
30 while(q--)
31 {
32     x1=read(),y1=read(),x2=read(),y2=read(),c=read();
33     b[x1][y1]+=c;
34     b[x1][y2+1]-=c;
35     b[x2+1][y1]-=c;
36     b[x2+1][y2+1]+=c;
37 }
38 for(int i=1;i<=n;i++)
39 {
40     for(int j=1;j<=m;j++)
41     {
42         a[i][j]=a[i-1][j]+a[i][j-1]-a[i-1][j-1]+b[i][j];
43         printf("%d ",a[i][j]);
44     }
45     printf("\n");
46 }
47 return 0;
48 }

```

注意事项

1. 前缀和，差分尽量使用快读
2. 涉及最大最小到时候min，max初值设为0，以免遗漏开头的
3. 前缀和左上角，差分右下角，两者互为逆运算，容斥定理可推公式
4. 前缀和区间快速求和，差分区间增减修改

字符串

KMP

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=1e5+10,M=1e6+10;
6  int n,m;
7  char p[N],s[M];
8  int ne[N]; //最长公共前缀后缀
9
10
11 int main()
12 {
13     cin>>n>>p+1>>m>>s+1;
14     //预处理 ne数组
15     for(int i=2,j=0;i<=n;i++) //从第二个开始处理，第一个肯定0啊
16     {
17         while(j&& p[i]!=p[j+1]) j=ne[j]; //j+1和i试探一下看一不一样，不一样就j=ne[i]
18         //直到开头

```

```

18         if(p[i]==p[j+1])j++;
19         ne[i]=j;
20     }
21     //kmp 匹配
22     for(int i=1,j=0;i<=m;i++)
23     {
24         while(j&& s[i]!=p[j+1])j=ne[j];
25         if(s[i]==p[j+1])j++;
26         if(j==n)
27         {
28             printf("%d ",i-n);
29             j=ne[j];
30         }
31     }
32     return 0;
33 }

```

区间操作

区间合并

DP

DP思考方式

状态表示

集合

1. 维度的确定是最少要用几个维度来唯一确定如：背包有容量和价值表状态，最原始两维度。最长子序列以i结尾，就只要一个。最长公共子序列两个序列，就要两个维度
2. 所有满足+（题目条件+状态表示的条件）+的集合

属性

1. max(开long long)
2. min（开long long)
3. 方案数（直接开unsigned long long防爆)
4. 具体方案（记录状态转移)

状态计算

划分

原则：补充不漏

1. 以i-1为倒数第二个（LIS)
2. 转移来源(数字三角)
3. 选i与不选i/选几个（背包)
4. a[i],b[j]是否包含在子序列当中

计算过程

要保证前面的已经计算好了，状态转移不能成环

背包

01背包

二维

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=1e4+10;
6
7
8  int n,m;
9  int v[N],w[N];
10 int f[N][N];
11
12
13 int main()
14 {
15     cin>>n>>m;
16     for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
17     //所有状态f[0~n][0~m]
18     //f[0][0~m]=0所以i就不从0开始了
19     for(int i=1;i<=n;i++)
20     {
21         for(int j=0;j<=m;j++)
22         {
23             f[i][j]=f[i-1][j]; //左边不含i，最大值就是f[i-1][j]
24             if(j>=v[i])f[i][j]=max(f[i][j],f[i-1][j-v[i]]+w[i]); //装得下v[i]
25             //才有这种情况，第一个就是左边最大值，第二个就是右边最大值，>=不要打成>
26         }
27     }
28     cout<<f[n][m]<<endl;
29     return 0;
30 }
```

一维

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=1e4+10;
5  int n,m;
6  int v[N],w[N];
7  int f[N]; //有时候要开long long不然会爆
8
9  int main()
10 {
11     cin>>n>>m;
12     for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
13
14     for(int i=1;i<=n;i++)
15         for(int j=m;j>=v[i];j--)
16             f[j]=max(f[j],f[j-v[i]]+w[i]);
17     cout<<f[m]<<endl;
```

```

18     return 0;
19
20 }

```

完全背包

二维

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=1e4+10;
6
7
8  int n,m;
9  int v[N],w[N];
10 int f[N][N];
11
12
13 int main()
14 {
15     cin>>n>>m;
16     for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
17
18     for(int i=1;i<=n;i++)
19         for(int j=0;j<=m;j++)
20             for(int k=0;k*v[i]<=j;k++)
21                 f[i][j]=max(f[i][j],f[i-1][j-v[i]*k]+w[i]*k);
22     cout<<f[n][m]<<endl;
23     return 0;
24

```

二维优化

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N = 1010;
4  int f[N][N];
5  int v[N],w[N];
6  int main()
7  {
8      int n,m;
9      cin>>n>>m;
10     for(int i = 1 ; i <= n ; i ++ )cin>>v[i]>>w[i];
11
12
13     for(int i = 1 ; i<=n ; i++)
14         for(int j = 0 ; j<=m ; j++)
15             {
16                 f[i][j] = f[i-1][j];
17                 if(j>=v[i])f[i][j]=max(f[i][j],f[i][j-v[i]]+w[i]);
18             }
19     cout<<f[n][m]<<endl;
20 }

```

一维优化

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=1e4+10;
5
6  int n,m;
7  int v[N],w[N]; //有时候要开long long不然会爆
8  int f[N];
9
10
11 int main()
12 {
13     cin>>n>>m;
14     for(int i=1;i<=n;i++)cin>>v[i]>>w[i];
15
16
17     for(int i=1;i<=n;i++)
18         for(int j=v[i];j<=m;j++)
19             f[j]=max(f[j],f[j-v[i]]+w[i]);
20     cout<<f[m]<<endl;
21     return 0;
22 }
```

多重背包

暴力朴素

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=1e4+10;
6
7
8  int n,m;
9  int v[N],w[N],s[i];
10 int f[N][N];
11
12
13 int main()
14 {
15     cin>>n>>m;
16     for(int i=1;i<=n;i++)cin>>v[i]>>w[i]>>s[i];
17
18
19     for(int i=1;i<=n;i++)
20         for(int j=0;j<=m;j++)
21             for(int k=0;k<=s[i]&& k*v[i]<=j;k++)
22                 f[i][j]=max(f[i][j],f[i][j-v[i]*k]+w[i]*k)
23     cout<<f[n][m]<<endl;
24     return 0;
25
26
27 }
```

二进制优化

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4
5  const int N=25000,M=2010;//N要拆出来所以1000*log2000
6
7
8  int n,m;
9  int v[N],w[N];
10 int f[N];
11 int main()
12 {
13     cin>>n>>m;
14     int cnt=0;
15     for(int i=1;i<=n;i++)
16     {
17         int a,b,s;//体积, 价值, 个数
18         cin>>a>>b>>s;
19         int k=1;
20         while(k<=s)
21         {
22             cnt++;
23             v[cnt]=a*k;//k个物品打包
24             w[cnt]=b*k;//k个物品打包
25             s-=k;
26             k*=2;
27         }
28         if(s>0)//补上c
29         {
30             cnt++;
31             v[cnt]=a*s;
32             w[cnt]=b*s;
33         }
34     }
35     //01背包
36     n=cnt;
37     for(int i=1;i<=n;i++)
38         for(int j=m;j>=v[i];j--)
39             f[j]=max(f[j],f[j-v[i]]+w[i]);
40
41     cout<<f[m]<<endl;
42     return 0;
43 }
```

分组背包

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=110;
5
```

```

6  int n,m;
7  int v[N][N],w[N][N],s[N]; //s表示第i组物品种类
8  int f[N];
9
10 int main()
11 {
12     cin>>n>>m; //n组物品, m容量
13     for(int i=1;i<=n;i++)
14     {
15         cin>>s[i];
16         for(int j=0;j<s[i];j++)
17             cin>>v[i][j]>>w[i][j];
18     }
19
20     for(int i=1;i<=n;i++)
21         for(int j=m;j>=0;j--) //i-1推i逆序
22             for(int k=0;k<s[i];k++) //有点像完全背包, k就是下标, 注意自己是0开始还是
1开始的。选第i组的第k件物品
23                 if(v[i][k]<=j)
24                     f[j]=max(f[j],f[j-v[i][k]]+w[i][k]);
25
26     cout<<f[m]<<endl;
27     return 0;
28 }

```

背包方案数

二维

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=1e4+10;
5  int w[N],f[N][N];
6  int main(){
7      int n,m;
8      scanf("%d%d",&n,&m);
9      for(int i=1;i<=n;i++)scanf("%d",&w[i]);
10     for(int i=0;i<=n;i++)f[i][0]=1; //从0开始
11
12     for(int i=1;i<=n;i++)
13         for(int j=1;j<=m;j++)
14         {
15             f[i][j]+=f[i-1][j];
16             if(j>=w[i])f[i][j]+=f[i-1][j-w[i]];
17         }
18
19     printf("%d",f[n][m]);
20     return 0;
21 }

```

一维

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4

```

```

5  const int N=1e4+10;
6  int w[N],f[N];
7  int main()
8  {
9      int n,m;
10     scanf("%d%d",&n,&m);
11     for(int i=1;i<=n;i++)scanf("%d",&w[i]);
12     f[0]=1;
13     for(int i=1;i<=n;i++)
14         for(int j=m;j>=w[i];j--)
15             f[j]+=f[j-w[i]];
16
17     printf("%d",f[m]);
18     return 0;
19 }

```

线性DP

数字三角

最长上升子序列 (LIS)

朴素DP $O(N^2)$

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=1e5+10;
4  int a[N],dp[N];
5  int n;
6  int main()
7  {
8      scanf("%d",&n);
9      for(int i=1;i<=n;i++)scanf("%d",&a[i]),dp[i]=1;
10     for(int i=1;i<=n;i++)
11         for(int j=1;j<i;j++)
12             if(a[i]>a[j])dp[i]=max(dp[i],dp[j]+1);
13
14     int ans=0;
15     for(int i=1;i<=n;i++)
16         ans=max(ans,dp[i]);
17     printf("%d",ans);
18     return 0;
19 }
20
21

```

记录最大上升子序列

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=1e5+10;
5  int f[N],a[N],g[N];
6  int main()
7  {

```

```

8     int n;
9     scanf("%d",&n);
10    for(int i=1;i<=n;i++)scanf("%d",&a[i]);
11    for(int i=1;i<=n;i++)
12    {
13        g[i]=0;
14        f[i]=1;
15        for(int j=1;j<i;j++)
16        {
17            if(a[j]<a[i])
18            {
19                if(f[i]<f[j]+1)
20                {
21                    f[i]=max(f[i],f[j]+1);
22                    g[i]=j;
23                }
24            }
25        }
26    }
27    int k=1;
28    for(int i=1;i<=n;i++)
29        if(f[k]<f[i])k=i;
30
31    printf("%d\n",f[k]);
32
33    for(int i=1,len=f[k];i<=len;i++)
34    {
35        printf("%d ",a[k]);
36        k=g[k];
37    }
38    return 0;
39 }

```

优化O(NlogN)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  const int N=1e5+10;
5  int n;
6  int a[N],q[N];//q[N]存长度为i的上升子序列的末尾元素最小值，LIS并不唯一
7  int main()
8  {
9      scanf("%d",&n);
10     for(int i=0;i<n;i++)scanf("%d",&a[i]);
11
12     int len=0;//最大长度
13     q[0]=-2e9;//保证一定存在小于a[i]的最大
14     for(int i=0;i<n;i++)
15     {
16         int l=0,r=len;
17         while(l<r)//二分找最后一个满足<=a[i]的点（在能接上的里面找最长的）
18         {
19             int mid=(l+r+1)>>1;
20             if(q[mid]<=a[i])l=mid;
21             else r=mid-1;
22         }

```

```

23     len=max(len,r+1);//r找的是接在那个后面
24     q[r+1]=a[i];
25 }
26 printf("%d\n",len);
27 }

```

最长公共子序列 (LCS)

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=1e3+10;
4  int n,m;
5  char a[N],b[N];
6  int dp[N][N];
7  int main()
8  {
9      scanf("%d%d",&n,&m);
10     scanf("%s %s",a+1,b+1);
11     for(int i=1;i<=n;i++)
12     {
13         for(int j=1;j<=m;j++)
14         {
15             dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
16             if(a[i]==b[j])dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
17         }
18     }
19     printf("%d",dp[n][m]);
20     return 0;
21 }

```

区间DP

记忆化搜索