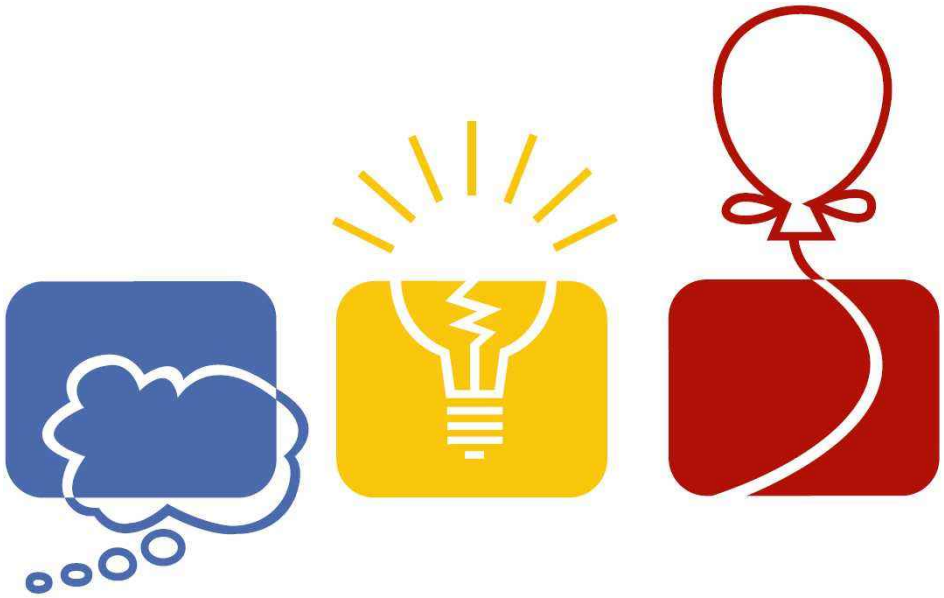

ACM

zjgsu

Mr.l

February 24, 2020

Contents

0 头文件

0.1 头文件

```
#pragma comment(linker, "/STACK:1024000000,1024000000")
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstring>
#include<vector>
#include<algorithm>
#include<sstream>
#include<map>
#include<queue>
#include<set>
#include<bitset>
#include<list>
using namespace std;

typedef pair<int, int> pii;
typedef long long ll;
typedef unsigned long long ull;
#define pw(k) ((1ll)<<(k))
const ull hash1 = 201326611;
const double eps = 1e-8;
const ll INF = 0x3f3f3f3f3f3f3f3f;
const int inf = 0x3f3f3f3f;
const ll mod = 1e9 + 7;
const int N = 2e6+10;
const int M = 12;
const int dif = 26;
const double PI = acos(-1.0);
ll Mod(ll x){ return (x%mod+mod)%mod;}
void BinaryBitset(int n) { cout << bitset<sizeof(int) * 4>(n) << endl; }
inline int getBinary(int x){int cnt=0; for(;x;x-=(x & (-x))) cnt++;return cnt;}

int main() {
#ifdef ACM_LOCAL
    freopen("./std.in", "r", stdin);
    //freopen("./std.out","w",stdout);
    auto start = clock();
#endif

#ifdef ACM_LOCAL
    auto end = clock();
    cerr << "Run Time: " << double(end - start) / CLOCKS_PER_SEC << "s" << endl;
#endif
}
```

0.2 读入模板

```
1、快读
struct ioss
{
#define endl '\n'
    static const int LEN = 20000000;
    char obuf[LEN], *oh = obuf;
    std::streambuf *fb;
```

```
8       ioss()
9       {
10          ios::sync_with_stdio(false);
11          cin.tie(NULL);
12          cout.tie(NULL);
13          fb = cout.rdbuf();
14      }
15      inline char gc()
16      {
17
18          static char buf[LEN], *s, *t, buf2[LEN];
19          return (s == t) && (t = (s = buf) + fread(buf, 1, LEN, stdin)), s == t ? -1 : *
        s++;
20      }
21      inline ioss &operator>>(long long &x)
22      {
23          static char ch, sgn, *p;
24          ch = gc(), sgn = 0;
25          for (; !isdigit(ch); ch = gc())
26          {
27              if (ch == -1)
28                  return *this;
29              sgn |= ch == '-';
30          }
31          for (x = 0; isdigit(ch); ch = gc())
32              x = x * 10 + (ch ^ '0');
33          sgn && (x = -x);
34          return *this;
35      }
36      inline ioss &operator>>(int &x)
37      {
38          static char ch, sgn, *p;
39          ch = gc(), sgn = 0;
40          for (; !isdigit(ch); ch = gc())
41          {
42              if (ch == -1)
43                  return *this;
44              sgn |= ch == '-';
45          }
46          for (x = 0; isdigit(ch); ch = gc())
47              x = x * 10 + (ch ^ '0');
48          sgn && (x = -x);
49          return *this;
50      }
51      inline ioss &operator>>(char &x)
52      {
53          static char ch;
54          for (; !isalpha(ch); ch = gc())
55          {
56              if (ch == -1)
57                  return *this;
58          }
59          x = ch;
60          return *this;
61      }
62      inline ioss &operator>>(string &x)
63      {
64          static char ch, *p, buf2[LEN];
65          for (; !isalpha(ch) && !isdigit(ch); ch = gc())
```

```
66              if (ch == -1)
67                  return *this;
68          p = buf2;
69          for (; isalpha(ch) || isdigit(ch); ch = gc())
70              *p = ch, p++;
71          *p = '\0';
72          x = buf2;
73          return *this;
74      }
75      inline ioss &operator<<(string &c)
76      {
77          for (auto &p : c)
78              this->operator<<(p);
79          return *this;
80      }
81      inline ioss &operator<<(const char *c)
82      {
83          while (*c != '\0')
84          {
85              this->operator<<(*c);
86              c++;
87          }
88          return *this;
89      }
90      inline ioss &operator<<(const char &c)
91      {
92          oh == obuf + LEN ? (fb->sputn(obuf, LEN), oh = obuf) : 0;
93          *oh++ = c;
94          return *this;
95      }
96      inline ioss &operator<<(int x)
97      {
98          static int buf[30], cnt;
99          if (x < 0)
100             this->operator<<('-'), x = -x;
101         if (x == 0)
102             this->operator<<('0');
103         for (cnt = 0; x; x /= 10)
104             buf[++cnt] = x % 10 | 48;
105         while (cnt)
106             this->operator<<((char)buf[cnt--]);
107         return *this;
108     }
109     inline ioss &operator<<(long long x)
110     {
111         static int buf[30], cnt;
112         if (x < 0)
113             this->operator<<('-'), x = -x;
114         if (x == 0)
115             this->operator<<('0');
116         for (cnt = 0; x; x /= 10)
117             buf[++cnt] = x % 10 | 48;
118         while (cnt)
119             this->operator<<((char)buf[cnt--]);
120         return *this;
121     }
122     ~ioss()
123     {
124         fb->sputn(obuf, oh - obuf);
```

```
    }
} io;

2、__int128读入

inline __int128 read(){
   int X=0,w=0; char ch=0;
   while(!isdigit(ch)) {w|=ch=='-';ch=getchar();}
   while(isdigit(ch)) X=(X<<3)+(X<<1)+(ch^48),ch=getchar();
   return w?-X:X;
}

void print(__int128 x){
    if (!x) return ;
    if (x < 0) putchar('-'),x = -x;
    print(x / 10);
    putchar(x % 10 + '0');
}

3、
template<class T>inline void read(T &res){
    char c;T flag=1;
    while((c=getchar())<'0'||c>'9')if(c=='-')flag=-1;res=c-'0';
    while((c=getchar())>='0'&&c<='9')res=res*10+c-'0';res*=flag;
}


4、整行读入，其中包含空格
//注意前后使用getchar()清除多余空字符
int i=0;
while((s[i]=getchar())!='\n') i++;
s[i]=0;


//注意前后使用cin.get()清除多余空字符
getline(cin,s);

在使用char参数或没有参数的情况下，get（）方法读取下一个字符，及时该字符是空格，制表符或换行符。get（
    char & ch）版本将输
入字符赋给其参数，而get（void）版本将输入字符转换为整型（通常为int）。然后将其返回。
```

1 字串符

1.1 KMP、E-KMP、Manacher

```
1.KMP模板

//求取循环节的基础
void getNext(){
    int i,j;
    j=nx[0]=-1;
    i=0;
    while(i<n){
        while(-1!=j && x[i]!=x[j]) j=nx[j];
        nx[++i]=++j;
    }
}

//消除循环节
void getNext(){
    int i,j;
    j=nx[0]=-1;
    i=0;
    while(i<m){
        while(-1!=j && x[i]!=x[j]) j=nx[j];
        if(x[++i]==x[++j]) nx[i] = nx[j];
        else nx[i]=j;
    }
}

//返回x在y中匹配次数，包含重叠
int KMP() {
    int i, j;
    int ans = 0;
    getNext(x, m, nx);
    i = j = 0;
    while (j < n) {
        while (i != -1 && x[i] != y[j])
            i = nx[i];
        i++; j++;
        if (i == m)
            ans++;
    }
    return ans;
}

2、扩展KMP
void pre_EKMP(){
    nx[0]=m;
    int j = 0;
    while(j+1<m&&x[j]==x[j+1]) j++;
    nx[1]=j;
    int k = 1;
    for(int i=2;i<m;i++){
        int p = nx[k]+k-1;
        int L = nx[i-k];
        if(i+L<p+1) nx[i]=L;
        else{
            j = max(0,p-i+1);
            while(i+j<m&&x[i+j]==x[j]) j++;
```

```
            nx[i]=j;
            k=i;
        }
    }
}

void EKMP(){
    pre_EKMP();
    int j = 0;
    while(j<n&&j<m&&x[j]==y[j]) j++;
    extend[0]=j;
    int k = 0;
    for(int i=1;i<n;i++){
        int p = extend[k]+k-1;
        int L = nx[i-k];
        if(i+L<p+1) extend[i]=L;
        else{
            j = max(0,p-i+1);
            while(i+j<n&&j<m&&y[i+j]==x[j]) j++;
            extend[i]=j;
            k=i;
        }
    }
}

3、Manacher
//内存开两倍
const int N = 110100;
char Ma[N];
int Mp[N],top;
void Manacher(char *s,int len){
    top=0;
    Ma[top++]='$';
    Ma[top++]='#';
    for(int i=0;i<len;i++){
        Ma[top++]=s[i];
        Ma[top++]='#';
    }
    Ma[top]=0; Mp[top]=0;
    int id=0,mx=0;
    for(int i=1;i<top;i++){
        Mp[i]=mx>i?min(Mp[2*id-i],mx-i):1;
        while(Ma[i+Mp[i]]==Ma[i-Mp[i]]) Mp[i]++;
        if(i+Mp[i]>mx){
            mx=i+Mp[i];
            id=i;
        }
    }
}
```

1.2 字典树

```
class tire{
public:
    int nx[N][dif],end[N];
    int root,tot;

    int newNode(){
```

```
        for(int i=0;i<26;i++) nx[tot][i]=-1;
        end[tot]=0;
        return tot++;
    }

    void init(){
        tot=0;
        root=newNode();
    }

    void insert(char *s,int id){
        int len=strlen(s);
        int now=root;
        for(int i=0;i<len;i++){
            int c=s[i]-'a';
            if(nx[now][c]==-1) nx[now][c]=newNode();
            now=nx[now][c];
        }
        end[now]=id;
    }

    int query(char *s){
        int len=strlen(s);
        int now=root;
        for(int i=0;i<len;i++){
            int c=s[i]-'a';
            if(nx[now][c]==-1) return -1;
            now=nx[now][c];
        }
        if(end[now]>0) return end[now];
        else return -1;
    }
}tr;
```

1.3 哈希

```
求hash值
    hash的特征就是不同的key（就是目标位置)对应的数据不同，所以将字符串转化为数字应该注意一对应， 避免哈希冲突(比如不同字符
串对应了同一个值，但是你的程序还是会判断它们是同一个字符串一般的字符串hash值求法(终于到正题了)给一个字符串从左到右枚举字符
串的每一位，每一个字母直接对应它的ASCII码(就变成int了)，对应好子就把每位加起来， 就输快的冲突子直接相加会冲突，例如ab和ba
,第二串后来的那个a和第一串的前面的a虽然一 个更老一个更年轻，但是它们的作用居然是一样的，这是不符合常识的(我是在说实话)
所以，为了使资质更老的a更显眼，可以在处理之后的那些后代的时候给它乘上一个数base显示它的不同。 如果考虑到每一个字符后面都有
后代的话，那么每处理一个后面的字符， 前面的祖宗们就都会乘上一一个数。容易看出，每个位置都比它后面那个位置多乘了-一次， 这样
就可以显示出各个位置的等级差距了，再结合之前的直接相加，就可以表示出来每一个不同的字符串 了，即:
    val ["abc"] = 'a'*base^2+'b'*base^1+'c'*base^0
    那么对于一个母串，怎么提取它[l, r ]中的hash值呢。我们已经知道了这个串从1到每个位置这一部分的hash值 ，这类似于前缀和，
即hash[r ]-hash[l-1],但是由于对于r位置的hash[r],它前面一部分(即被它包含在内的hash[ 1]部分)被多乘了许多次base，减的时候
应该给hash[ I ]他乘上(换个说法:求出hash[l-1 ]之后，继续向后面走，每走一步都会hash[L-1 ]乘上base，直求到hash[ r ]时已经
```

```
乘了(r-I+1)个base了， 实际上hash[r]=hash[l,r]+hash[1-1]*base^(r-l+1)所以答案应该是(多乘了的次数[l,r]区间长度)
val[i,r]=hash[r]-hash[ 1-1 ]*base^(r-l+1)最后，因为乘的base-般很大，所以乘多了容易爆，要取模，为了避免麻烦，一般使用
unsigned long long
Qhash如何支持单点修改?
A可以用线段树维护
要用线段树维护要资瓷区间台并>
hash=左子树hash*(base^右子树size)+右子树hash

struct HASH{
    ull hash1;
    ull p[N],ha[N];
    void init(int n){
        hash1=201326611;//233,50331653
        p[0]=1; ha[0]=0;
        for(int i=1;i<=n;i++)
            p[i]=p[i-1]*hash1;
    }
    //传指针要从s开始，不用s+1
    void build(char *s,int x,int n){
        for(int i=x;i<=n;i++) ha[i]=ha[i-1]*hash1+s[i];
    }
    ull getha(int l,int r){
        return ha[r]-ha[l-1]*p[r-l+1];
    }
    int query(int x,int y){
        int right=n-max(x,y)+1,left=1;
        while(left<=right){
            int mid=(left+right)>>1;
            if(getha(x,x+mid-1)==getha(y,y+mid-1)) left=mid+1;
            else right=mid-1;
        }
        return right;
    }
}hs;

//将可以通过此类方法将二维、三维等压缩到一维ha数组中
int ha[10000007];
ll seed=309989,mod = 9989783;

inline int gethash(ll x,ll y){
    int t=(x*seed+y)%mod;
    return t;
}
```

1.4 AC 自动机

```
1、查询一个串可以匹配多少个串
int n;
char s[M];

//ac自动机中dif看具体字符串出现的种类数，而N为字符串数*字符串长度
//nx[i][j]表示i节点若下一个字符为j时转向的节点，fail[i]表示i节点失配时指向的节点
//end[i]表示i节点所包含的状态,tot所表示节点范围为[0,tot-1],root=0
class tree {
public:
    int nx[N][dif], fail[N];
```

```
11      int end[N];
12      int root, tot;
13
14      int newNode() {
15          for (int i = 0; i < 26; i++)
16              nx[tot][i] = -1;
17          end[tot] = 0;
18          return tot++;
19      }
20
21      void init() {
22          tot = 0;
23          root = newNode();
24      }
25
26      //插入字母注意调整
27      void insert(char* s) {
28          int len = strlen(s);
29          int now = root;
30          for (int i = 0; i < len; i++) {
31              int id = s[i] - 'a';
32              if (nx[now][id] == -1)
33                  nx[now][id] = newNode();
34              now = nx[now][id];
35          }
36          end[now]++;
37      }
38
39      void build() {
40          queue<int>q;
41          fail[root] = root;
42          for (int i = 0; i < 26; i++) {
43              if (nx[root][i] == -1) {
44                  nx[root][i] = root;
45              }
46              else {
47                  fail[nx[root][i]] = root;
48                  q.push(nx[root][i]);
49              }
50          }
51          while (!q.empty()) {
52              int now = q.front();
53              q.pop();
54              //if (end[fail[now]]) end[now] = 1; 看情况加入
55              for (int i = 0; i < 26; i++) {
56                  if (nx[now][i] == -1) {
57                      nx[now][i] = nx[fail[now]][i];
58                  }
59                  else {
60                      fail[nx[now][i]] = nx[fail[now]][i];
61                      q.push(nx[now][i]);
62                  }
63              }
64          }
65      }
66
67      //查询存在
68      int query(char* s) {
69          int res = 0,len = strlen(s);
```

```
        int now = root;
        for (int i = 0; i < len; i++) {
            now = nx[now][s[i] - 'a'];
            int tmp = now;
            while (tmp != root&&end[tmp]!=0) {
                res += end[tmp];
                end[tmp] = 0;
                tmp = fail[tmp];
            }
        }
        return res;
    }
}ac;

int main() {
    int t;
    scanf("%d",&t);
    while(t--){
        ac.init();
        scanf("%d",&n);
        for(int i=1;i<=n;i++){
            scanf("%s",s); ac.insert(s);
        }
        ac.build();
        scanf("%s",s);
        printf("%d\n",ac.query(s));
    }
}

2、查询一个字符串中子串出现可重叠子串个数
int query(char s[]) {
   int res = 0;
   int len = strlen(s);
   int now = root;
   for (int i = 0; i < len; i++) {
        now = nx[now][s[i]];
        int tmp = now;
        while (tmp != root) {
              if(end[tmp]) v[end[tmp]]++;
              tmp = fail[tmp];
        }
   }
   return res;
}

3、有m(m<=10)个长度不超过10的只包括AGCT字符串,现在求产生n(n<=1e9)长度的字符串,并且不包括m个字符串,
    求种类数
通过ac自动机产生关系矩阵.
build函数中要加入: if (end[fail[now]]) end[now] = 1;
void pre_mat() {
    memset(a.arr, 0, sizeof a.arr);
    for (int i = 0; i < tot; i++) {
        if (end[i]) continue;
        for (int j = 0; j < 4; j++) {
            int k = nx[i][j];
            if (!end[k]) a.arr[i][k]++;
        }
    }
}
```

或当n小的时候使用dp即可实现

```
int n,m;
char s[N];

class tree {
public:
    int nx[N][2], fail[N];
    int end[N];
    int root, tot;

    int newNode() {
        for (int i = 0; i < 2; i++)
            nx[tot][i] = -1;
        end[tot] = 0;
        return tot++;
    }

    void init() {
        tot = 0;
        root = newNode();
    }

    void insert(char* s) {
        int len = strlen(s);
        int now = root;
        for (int i = 0; i < len; i++) {
            int id = s[i]-'0';
            if (nx[now][id] == -1)
                nx[now][id] = newNode();
            now = nx[now][id];
        }
        end[now]=1;
    }

    void build() {
        queue<int>q;
        fail[root] = root;
        for (int i = 0; i < 2; i++) {
            if (nx[root][i] == -1) {
                nx[root][i] = root;
            }
            else {
                fail[nx[root][i]] = root;
                q.push(nx[root][i]);
            }
        }
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            if (end[fail[now]]) end[now] = 1;
            for (int i = 0; i < 2; i++) {
                if (nx[now][i] == -1) {
                    nx[now][i] = nx[fail[now]][i];
                }
                else {
                    fail[nx[now][i]] = nx[fail[now]][i];
                    q.push(nx[now][i]);
                }
```

```
            }
        }
    }
}ac;

ll dp[50][N];

int main() {
    int t;
    scanf("%d",&t);
    while(t--){
        scanf("%d%d%s",&n,&m,s);
        ac.init();
        ac.insert(s);
        for(int i=0;i<n;i++){
            if(s[i]=='1') s[i]='0';
            else s[i]='1';
            ac.insert(s);
            if(s[i]=='0') s[i]='1';
            else s[i]='0';
        }
        ac.build();
        for (int i = 0; i <= m; i++) {
            for (int j = 0; j < ac.tot; j++) {
                dp[i][j] = 0;
            }
        }
        dp[0][0] = 1;
        for(int i=1;i<=m;i++){
            for(int j=0;j<ac.tot;j++){
                if(ac.end[j]) continue;
                for(int k=0;k<2;k++){
                    int x=ac.nx[j][k];
                    if(!ac.end[x]){
                        dp[i][x]+=dp[i-1][j];
                    }
                }
            }
        }
        ll ret=0;
        for(int i=0;i<ac.tot;i++) ret+=dp[m][i];
        printf("%lld\n",pw(m)-ret);
    }
}
```

4、给出一个文本串和n次查询，每次查询给出一个模式串和相应标记op——若op为0说明查询文本串时模式串允许重叠，若op为
1说明查询时模式串不能重叠。对每次查询，输出当前模式串在文本串中出现的个数。

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

typedef long long ll;
const int N = 1e5 + 10;
char str[N];
int n;
char s[2][N][8];
int p[2][N];
int cnt, cnt2;
int v[N],len,vis[N];
```

```
int l[N];
int vis2[N*10];


class tree {
public:
    int nx[600005][26], fail[600005];
    vector<int>end[600005];
    int root, tot;

    int newNode() {
        for (int i = 0; i < 26; i++)
            nx[tot][i] = -1;
        end[tot].clear();
        vis2[tot] = 0;
        return tot++;
    }

    void init() {
        tot = 0;
        root = newNode();
    }

    void insert(char s[8], int x) {
        int len = strlen(s);
        int now = root;
        for (int i = 0; i < len; i++) {
            int id = s[i]-'a';
            if (nx[now][id] == -1)
                nx[now][id] = newNode();
            now = nx[now][id];
        }
        end[now].push_back(x);
    }

    void build() {
        queue<int>q;
        fail[root] = root;
        for (int i = 0; i < 26; i++) {
            if (nx[root][i] == -1) {
                nx[root][i] = root;
            }
            else {
                fail[nx[root][i]] = root;
                q.push(nx[root][i]);
            }
        }
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            for (int i = 0; i < 26; i++) {
                if (nx[now][i] == -1) {
                    nx[now][i] = nx[fail[now]][i];
                }
                else {
                    fail[nx[now][i]] = nx[fail[now]][i];
                    q.push(nx[now][i]);
                }
            }
```

```
        }
    }

    void query1() {
        int now = root;
        for (int i = 0; i < len; i++) {
            int id = str[i] - 'a';
            now = nx[now][id];
            int tmp = now;
            while (tmp != root) {
                if (end[tmp].size()) {
                    vis2[tmp]++;
                }
                tmp = fail[tmp];
            }
        }
        for (int i = 0; i < tot; i++) {
            if (vis2[i]) {
                for (int j = 0; j < end[i].size(); j++) {
                    v[end[i][j]]+=vis2[i];
                }
            }
        }
    }
    void query2() {
        int now = root; int step = 0;
        for (int i = 0; i < len; i++) {
            int id = str[i] - 'a'; step++;
            now = nx[now][id];
            int tmp = now;
            while (tmp != root) {
                if (end[tmp].size() && (step - vis[tmp]) >= l[end[tmp][0]]) {
                    vis[tmp] = step;
                    vis2[tmp]++;
                }
                tmp = fail[tmp];
            }
        }
        for (int i = 0; i < tot; i++) {
            if (vis2[i]) {
                for (int j = 0; j < end[i].size(); j++) {
                    v[end[i][j]] += vis2[i];
                }
            }
        }
    }
}ac;

int main() {
    int t=1,pos;
    while(scanf("%s",str)==1){
        len = strlen(str);
        cnt = cnt2 = 0;
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &pos); v[i] = vis[i] = l[i] = 0;
            if (pos == 0) {
                scanf("%s", s[pos][++cnt]);
                p[pos][cnt] = i;
```

```
            }
            else {
                scanf("%s", s[pos][++cnt2]);
                l[i] = strlen(s[pos][cnt2]);
                p[pos][cnt2] = i;
            }
        }
        ac.init();
        for (int i = 1; i <= cnt; i++)
            ac.insert(s[0][i], p[0][i]);
        ac.build();
        ac.query1();
        ac.init();
        for (int i = 1; i <= cnt2; i++)
            ac.insert(s[1][i], p[1][i]);
        ac.build();
        ac.query2();
        printf("Case %d\n",t++);
        for (int i = 1; i <= n; i++)
            printf("%d\n", v[i]);
        printf("\n");
    }
}

#include <cstdio>
#include <cstring>
#include <queue>
#include <algorithm>
#define MAXN 600000+10
#define INF 0x3f3f3f3f
using namespace std;
int ans[MAXN][2];
int node[100000+10];//记录串在Trie中的结束点
int n;
int op[100000+10];
struct Trie
{
    int next[MAXN][26], fail[MAXN];
    int pos[MAXN];//记录当前节点的字符在模式串的位置
    int last[MAXN];//记录当前节点上一个匹配的位置
    int L, root;
    int newnode()
    {
        for(int i = 0; i < 26; i++)
            next[L][i] = -1;
        //End[L++] = 0;
        pos[L++] = 0;//这里忘写了，MLE到死。。。
        return L-1;
    }
    void init()
    {
        L = 0;
        root = newnode();
    }
    void Insert(char *s, int id)
    {
        int now = root;
        for(int i = 0; s[i]; i++)
        {
```

```
422             if(next[now][s[i]-'a'] == -1)
423                 next[now][s[i]-'a'] = newnode();
424             now = next[now][s[i]-'a'];
425             pos[now] = i+1;
426         }
427         node[id] = now;//记录串结束点
428     }
429     void Build()
430     {
431         queue<int> Q;
432         fail[root] = root;
433         for(int i = 0; i < 26; i++)
434         {
435             if(next[root][i] == -1)
436                 next[root][i] = root;
437             else
438             {
439                 fail[next[root][i]] = root;
440                 Q.push(next[root][i]);
441             }
442         }
443         while(!Q.empty())
444         {
445             int now = Q.front();
446             Q.pop();
447             for(int i = 0; i < 26; i++)
448             {
449                 if(next[now][i] == -1)
450                     next[now][i] = next[fail[now]][i];
451                 else
452                 {
453                     fail[next[now][i]] = next[fail[now]][i];
454                     Q.push(next[now][i]);
455                 }
456             }
457         }
458     }
459     void solve(char *s)
460     {
461         memset(last, -1, sizeof(last));
462         memset(ans, 0, sizeof(ans));
463         int len = strlen(s);
464         int now = root;
465         for(int i = 0; i < len; i++)
466         {
467             now = next[now][s[i]-'a'];
468             int temp = now;
469             while(temp != root)
470             {
471                 ans[temp][0]++;
472                 if(i - last[temp] >= pos[temp])
473                 {
474                     ans[temp][1]++;
475                     last[temp] = i;
476                 }
477                 temp = fail[temp];
478             }
479         }
480     }
```

```
};
Trie ac;
char str[100000+10];
char s[10];
int main()
{
    int k = 1;
    while(scanf("%s", str) != EOF)
    {
        ac.init(); scanf("%d", &n);
        for(int i = 0; i < n; i++)
        {
            scanf("%d%s", &op[i], s);
            ac.Insert(s, i);
        }
        ac.Build(); ac.solve(str);
        printf("Case %d\n", k++);
        for(int i = 0; i < n; i++)
            printf("%d\n", ans[node[i]][op[i]]);
        printf("\n");
    }
    return 0;
}
```

1.5 回文自动机

```
裸模板
1.len[i]表示编号为i的节点表示的回文串的长度（一个节点表示一个回文串）
2.next[i][c]表示编号为i的节点表示的回文串在两边添加字符c以后变成的回文串的编号（和字典树类似）。
3.fail[i]表示节点i失配以后跳转不等于自身的节点i表示的回文串的最长后缀回文串（和AC自动机类似）。
4.cnt[i]表示节点i表示的本质不同的串的个数（建树时求出的不是完全的，最后count()函数跑一遍以后才是正确
    的）
5.num[i]表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数。
6.last指向新添加一个字母后所形成的最长回文串表示的节点,便于下次insert。
7.s[i]表示第i次添加的字符（一开始设s[0] = -1（可以是任意一个在串s中不会出现的字符））。
8.p表示添加的节点个数。
9.tot表示添加的字符个数。
10.偶子树根节点为0，奇子树根节点为1，fail[0]指向1,len[0]=0,len[1]=-1,now的上一个节点为cur
class PalindromicTree{
public:
    int nx[N][dif],fail[N],len[N],cnt[N],num[N];
    int tot,p,last,s[N];
    int newnode(int l){
        memset(nx[p],0, sizeof(nx[p]));
        len[p]=l;
        cnt[p]=num[p]=0;
        return p++;
    }
    void init(){
        tot=p=last=0;
        s[0]=-1,fail[0]=1;
        newnode(0);
        newnode(-1);
    }
    int getfail(int x){
        while(s[tot-len[x]-1]!=s[tot])
            x=fail[x];
        return x;
```

```
    }
    void insert(int x){
        s[++tot]=x;
        int cur = getfail(last);
        int now = nx[cur][x];
        if(!now){
            now = newnode(len[cur]+2);
            fail[now]=nx[getfail(fail[cur])][x];
            nx[cur][x]=now;
            num[now]=num[fail[now]]+1;
        }
        last=nx[cur][x];
        cnt[last]++;
    }
    void makecnt(){
        for(int i=p-1;i>=2;i--)
            cnt[fail[i]]+=cnt[i];
    }
}pt;

可以双向增加,查询本质不同回文串个数,已经生成回文串个数
const int dif = 26;
int n, q;
int op;
char str[5];


class PalindromicTree{
public:
    int nx[N][dif],fail[N],len[N],num[N];
    int tot[2],p,last[2],s[N];
    int newnode(int l){
        memset(nx[p],0, sizeof(nx[p]));
        len[p]=l;
        num[p]=0;
        return p++;
    }
    void init(int x){
        memset(s,-1, sizeof(s));
        last[0]=last[1]=p=0;
        tot[0]=x;tot[1]=x-1;
        fail[0]=fail[1]=1;
        newnode(0);
        newnode(-1);
    }
    int getfail(int x,int tag){
        if(!tag){
            while(s[tot[tag]+len[x]+1]!=s[tot[tag]])
                x=fail[x];
        }else{
            while(s[tot[tag]-len[x]-1]!=s[tot[tag]])
                x=fail[x];
        }
        return x;
    }
    int insert(int x,int tag){
        if(!tag)
            s[--tot[0]]=x;
        else
```

```
            s[++tot[1]]=x;
        int cur = getfail(last[tag],tag);
        int now = nx[cur][x];
        if(!now){
            now = newnode(len[cur]+2);
            fail[now]=nx[getfail(fail[cur],tag)][x];
            nx[cur][x]=now;
            num[now]+=num[fail[now]]+1;
        }
        last[tag]=nx[cur][x];
        if(len[last[tag]]==tot[1]-tot[0]+1)
            last[tag^1]=last[tag];
        return num[last[tag]];
    }
}pt;


int main() {
    while(scanf("%d",&q)==1){
        pt.init(q);
        ll ans=0;
        while(q--){
            scanf("%d",&op);
            if(op==1){
                scanf("%s",str);
                ans+=pt.insert(str[0]-'a',0);
            }else if(op==2){
                scanf("%s",str);
                ans+=pt.insert(str[0]-'a',1);
            }else if(op==3){
                printf("%d\n",pt.p-2);
            }else{
                printf("%lld\n",ans);
            }
        }
    }
    return 0;
}

3、使用vector优化内存
class PalindromicTree{
public:
    vector<pii>nx[N];
    int fail[N],len[N],num[N];
    int tot,p,last,s[N];
    int newnode(int l){
        nx[p].clear();
        len[p]=l;
        num[p]=0;
        return p++;
    }
    void init(){
        tot=p=last=0;
        s[0]=-1,fail[0]=1;
        newnode(0);
        newnode(-1);
    }
    int getfail(int x){
        while(s[tot-len[x]-1]!=s[tot])
```

```
            x=fail[x];
        return x;
    }

    int is_exist(int p,int c){
        for(auto t:nx[p]){
            if(t.first==c)
                return t.second;
        }
        return 0;
    }

    void insert(int x){
        s[++tot]=x;
        int cur = getfail(last);
        int now = is_exist(cur,x);
        if(!now){
            now = newnode(len[cur]+2);
            fail[now]=is_exist(getfail(fail[cur]),x);
            nx[cur].push_back(make_pair(x,now));
            num[now]=num[fail[now]]+1;
        }
        last=now;
    }
}pt;
```

1.6 后缀数组、后缀自动机

后缀数组模板

```
int n,m;
int sa[N],c[N],wa[N],wb[N];
int rk[N],height[N];
//sa[i]表示排名为i的后缀的起始位置的下标，rk[i]表示起始位置的下标为i的后缀的排名
//c[i]表示桶，x[i]是第i个元素的第一关键字,y[i]表示第二关键字排名为i的数，第一关键字的位置
//height[i]为LCP(i,i-1)，1<i<=n，显然height[1]=0;
//设h[i]=height[rk[i]]，同样的，height[i]=h[sa[i]]；则有h[i]>=h[i-1]-1;

void DA(int *s,int n,int m){
    //如果多组数据,x需要初始化
    for(int i=0;i<=m;i++) c[i]=wa[i]=0;
    int *x=wa,*y=wb;
    for(int i=1;i<=n;i++) ++c[x[i]=s[i]];
    for(int i=2;i<=m;i++) c[i]+=c[i - 1];
    for(int i=n;i>=1;i--) sa[c[x[i]]--] = i;
    for(int k=1;k<=n;k<<=1){
        int p = 0;
        for(int i=n-k+1;i<=n;i++) y[++p]=i;
        for(int i=1;i<=n;i++) if(sa[i]>k) y[++p] = sa[i]-k;
        for(int i=0;i<=m;i++) c[i]=0;
        for(int i=1;i<=n;i++) ++c[x[i]];
        for(int i=2;i<=m;i++) c[i]+=c[i-1];
        for(int i=n;i>=1;i--) sa[c[x[y[i]]]--]=y[i];
        swap(x,y);
        x[sa[1]]=1;
        p=1;
        for(int i=2;i<=n;i++) x[sa[i]]=(y[sa[i]]==y[sa[i-1]]&&y[sa[i]+k]==y[sa[i-1]+k])
     ? p : ++p;
        if(p==n) break;
```

```
        m=p;
    }
    int k=0;
    for (int i=1; i<=n; ++i) rk[sa[i]]=i;
    for (int i=1; i<=n; ++i) {
        if (rk[i]==1) continue;//第一名height为0
        if (k) --k;//h[i]>=h[i-1]-1;
        int j=sa[rk[i]-1];
        while (j+k<=n && i+k<=n && s[i+k]==s[j+k]) ++k;
        height[rk[i]]=k;//h[i]=height[rk[i]];
    }
}

int main() {
    scanf("%s",s+1);
    int n = strlen(s+1);
    //注意下标从s开始，而不是s+1
    DA(s,n,127);
    for(int i=1;i<=n;i++) printf("%d ",sa[i]);
}
```

后缀数组dc3:

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
#define G(x) ((x) < tb ? (x) * 3 + 1 : ((x) - tb) * 3 + 2)
using namespace std;
//开三倍空间
const int N = 3000005;
int wa[N], wb[N], wss[N], wv[N], sa[N];
int rnk[N], height[N], s[N];
char str[N];
//sa和rnk实际上是从[0,len]的所有值，而s[len]=0,height正常


int c0(int *r, int a, int b) {
    return r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2];
}

int c12(int k, int *r, int a, int b) {
    if (k == 2)
        return r[a] < r[b] || r[a] == r[b] && c12(1, r, a + 1, b + 1);
    return r[a] < r[b] || r[a] == r[b] && wv[a + 1] < wv[b + 1];
}

void Rsort(int *r, int *a, int *b, int n, int m) {
    for (int i = 0; i < n; i++) wv[i] = r[a[i]];
    for (int i = 0; i < m; i++) wss[i] = 0;
    for (int i = 0; i < n; i++) wss[wv[i]]++;
    for (int i = 1; i < m; i++) wss[i] += wss[i - 1];
    for (int i = n - 1; i >= 0; i--) b[--wss[wv[i]]] = a[i];
}

void dc3(int *r, int *sa, int n, int m) {
    int i, j, *rn = r + n, *san = sa + n, ta = 0, tb = (n + 1) / 3, tbc = 0, p;
    r[n] = r[n + 1] = 0;
    for (i = 0; i < n; i++) if (i % 3 != 0) wa[tbc++] = i;
    Rsort(r + 2, wa, wb, tbc, m);
```

```
    Rsort(r + 1, wb, wa, tbc, m);
    Rsort(r, wa, wb, tbc, m);
    for (p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++)
        rn[F(wb[i])] = c0(r, wb[i - 1], wb[i]) ? p - 1 : p++;
    if (p < tbc) dc3(rn, san, tbc, p);
    else for (i = 0; i < tbc; i++) san[rn[i]] = i;
    for (i = 0; i < tbc; i++) if (san[i] < tb) wb[ta++] = san[i] * 3;
    if (n % 3 == 1) wb[ta++] = n - 1;
    Rsort(r, wb, wa, ta, m);
    for (i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;
    for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++)
        sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j++];
    for (; i < ta; p++) sa[p] = wa[i++];
    for (; j < tbc; p++) sa[p] = wb[j++];
}

void calheight(int *r, int *sa, int n) {
    int i, j, k = 0;
    for (i = 1; i <= n; i++) rnk[sa[i]] = i;
    for (i = 0; i < n; height[rnk[i++]] = k)
        for (k ? k-- : 0, j = sa[rnk[i] - 1]; r[i + k] == r[j + k]; k++);
}

int main() {
    while (scanf("%s", str) == 1 && str[0] != '.') {
        int len = strlen(str);
        for (int i = 0; i < len; i++)
            s[i] = str[i] - 'a' + 1;
        s[len] = 0;
        dc3(s, sa, len + 1, 105);
        calheight(s, sa, len);
        int aa = len - height[rnk[0]];
        int ans = 1;
        if(len % aa == 0) {
            ans = len / aa;
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

后缀自动机:

```
//凡是和后缀自动机相关的数组都必须开2倍
int n;
char s[N];
int sz[N<<1],c[N<<1],rk[N<<1];

class SuffixAutoMaton{
public:
    int last,tot;
    int nx[N<<1][dif],fa[N<<1],len[N<<1];
    void init(){
        last=tot=1;
        fa[1]=len[1]=0;
        memset(nx[1],0, sizeof(nx[1]));
    }
    inline void insert(int c){
        int p=last,np=++tot;
        memset(nx[np],0, sizeof(nx[np]));
```

```
148         last=np; len[np]=len[p]+1;
149         for(;p&&!nx[p][c];p=fa[p]) nx[p][c]=np;
150         if(!p) fa[np]=1;
151         else{
152             int q=nx[p][c];
153             if(len[p]+1==len[q]) fa[np]=q;
154             else{
155                 int nq=++tot;
156                 len[nq]=len[p]+1;
157                 memcpy(nx[nq],nx[q], sizeof(nx[q]));
158                 fa[nq]=fa[q]; fa[q]=fa[np]=nq;
159                 for(;nx[p][c]==q;p=fa[p])
160                     nx[p][c]=nq;
161             }
162         }
163         sz[np]=1;
164     }
165     //求出S的所有出现次数不为1的子串的出现次数乘上该子串长度的最大值。
166     ll query(){
167         ll ans=0;
168         for(int i=1;i<=tot;i++) c[len[i]]++;
169         for(int i=1;i<=tot;i++) c[i]+=c[i-1];
170         for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
171         for(int i=tot;i>=1;i--){
172             int p=rk[i];
173             sz[fa[p]]+=sz[p];
174             if(sz[p]>1){
175                ans=max(ans,(ll)sz[p]*len[p]);
176             }
177         }
178         return ans;
179     }
180 }SAM;
181
182 //set优化nx
183 class SuffixAutoMaton{
184 public:
185     int last,tot;
186     set<pii>nx[N<<1];
187     int fa[N<<1],len[N<<1];
188     ll ret;
189     void init(){
190         last=tot=1;
191         fa[1]=len[1]=0; nx[1].clear(); ret=0;
192     }
193     inline void insert(int c){
194         int p=last,np=++tot;
195         nx[np].clear();
196         last=np; len[np]=len[p]+1;
197         for(;p;p=fa[p]) {
198             auto x=nx[p].lower_bound(make_pair(c,0));
199             if(x==nx[p].end()||(*x).first!=c){
200                 nx[p].insert(make_pair(c,np));
201             }else break;
202         }
203         if(!p) fa[np]=1;
204         else{
205             int q=(*nx[p].lower_bound(make_pair(c,0))).second;
206             if(len[p]+1==len[q]) fa[np]=q;
```

```
            else{
                int nq=++tot;
                len[nq]=len[p]+1;
                nx[nq]=nx[q];
                fa[nq]=fa[q]; fa[q]=fa[np]=nq;
                for(;;p=fa[p]) {
                    auto x=nx[p].lower_bound(make_pair(c,0));
                    if((*x).second!=q){
                        break;
                    }else{
                        nx[p].erase(x);
                        nx[p].insert(make_pair(c,nq));
                    }
                }
            }
        }
    }
}SAM;

//set优化vector
vector<pii>::iterator ite;

class SuffixAutoMaton{
public:
    int last,tot;
    vector<pii>nx[N<<1];
    int fa[N<<1],len[N<<1];
    ll ret;
    void init(){
        last=tot=1;
        fa[1]=len[1]=0; nx[1].clear(); ret=0;
    }

    int isOK(int p,int c){
        for(auto x:nx[p]){
            if(x.first==c) return x.second;
        }
        return 0;
    }

    void Del(int p,int c){
        for(ite=nx[p].begin();ite!=nx[p].end();ite++){
            if((*ite).first==c){
                nx[p].erase(ite);return;
            }
        }
    }

    inline void insert(int c){
        int p=last,np=++tot;
        nx[np].clear();
        last=np; len[np]=len[p]+1;
        for(;!isOK(p,c);p=fa[p]) nx[p].push_back(make_pair(c,np));
        if(!p) fa[np]=1;
        else{
            int q=isOK(p,c);
            if(len[p]+1==len[q]) fa[np]=q;
            else{
                int nq=++tot;
```

```
                len[nq]=len[p]+1;
                nx[nq]=nx[q];
                fa[nq]=fa[q]; fa[q]=fa[np]=nq;
                for(;isOK(p,c)==q;p=fa[p]) {
                    Del(p,c);
                    nx[p].push_back(make_pair(c,nq));
                }
            }
        }
        ret+=len[np]-len[fa[np]];
    }
}SAM;

//广义后缀自动机

1、直接建立新节点
给定两个字符串，求出在两个字符串中各取出一个子串使得这两个子串相同的方案数。两个方案不同当且仅当这两个子串中有一个位置不同。
//每次加入新字符串需要将last=1,此模板有时候有节点表示完全相同的东西len[fa[i]]==len[i]如果直接进行基数排序会wa SAM能直接基数
//排序代表拓扑序成立的条件是len[i]严格大于len[fa[i]]，（比如ab,abc）因此不能使用基数排序，而是建立fail树进行答案计数。
class SuffixAutoMaton{
public:
    int last,tot;
    int nx[N<<1][dif],fa[N<<1],len[N<<1];
    ll sz[2][N<<1];
    void init(){
        last=tot=1;
        fa[1]=len[1]=0;
        memset(nx[1],0, sizeof(nx[1]));
    }
    inline void insert(int c,int op){
        int p=last,np=++tot;
        memset(nx[np],0,sizeof(nx[np]));
        last=np; len[np]=len[p]+1;
        for(;p&&!nx[p][c];p=fa[p]) nx[p][c]=np;
        if(!p)
            fa[np]=1;
        else{
            int q=nx[p][c];
            if(len[p]+1==len[q]) fa[np]=q;
            else{
                int nq=++tot;
                len[nq]=len[p]+1;
                memcpy(nx[nq],nx[q], sizeof(nx[q]));
                fa[nq]=fa[q]; fa[q]=fa[np]=nq;
                for(;nx[p][c]==q;p=fa[p])
                    nx[p][c]=nq;
            }
        }
        sz[op][np]=1;
    }

    int head[N<<1],cnt;
    struct Edge{
        int to,nx;
    }e[N];

```

```
322     inline void addedge(int a,int b){
323         e[cnt]=(Edge){b,head[a]}; head[a]=cnt++;
324     }
325 
326     ll ret;
327 
328     void dfs(int u){
329         for(int i=head[u];i;i=e[i].nx){
330             int v=e[i].to;
331             dfs(v);
332             sz[0][u]+=sz[0][v]; sz[1][u]+=sz[1][v];
333             ret+=(len[v]-len[u])*sz[0][v]*sz[1][v];
334         }
335     }
336 
337     void query(){
338         memset(head,0, sizeof(int)*(tot+1));cnt=1;
339         for(int i=2;i<=tot;i++) addedge(fa[i],i);
340         ret=0; dfs(1);
341         printf("%lld\n",ret);
342     }
343 }SAM;
344 
345 //若想直接使用基数排序可以在main函数中进行一个小操作
346 //通过画{ab,abc}可知，节点(2,4)和节点(3,5)其实代表一样的东西，因此直接在main函数将p进行转移
347 //使得基数排序时不会出错
348 int main(){
349     SAM.init();
350     scanf("%s",s);
351     n=strlen(s);
352     int p=1;
353     for(int i=0;i<n;i++)
354         SAM.insert(s[i]-'a'),p=SAM.nx[p][s[i]-'a'],SAM.sz[0][p]++;
355     SAM.last=p=1;
356     scanf("%s",s);
357     n=strlen(s);
358     for(int i=0;i<n;i++)
359         SAM.insert(s[i]-'a'),p=SAM.nx[p][s[i]-'a'],SAM.sz[1][p]++;
360     SAM.calc();
361 }
362 
363 
364 //选择性建立新节点
365 class SuffixAutoMaton{
366 public:
367     int last,tot;
368     int nx[N<<1][26],fa[N<<1],len[N<<1];
369     int sz[2][N<<1],rk[N<<1],c[N<<1];
370     void init(){
371         last=tot=1;
372         fa[1]=len[1]=0;
373         memset(nx[1],0, sizeof(nx[1]));
374     }
375     //新字符串插入时需要提前将last设置为1
376     inline void insert(int c){
377         int p=last;
378         if(nx[p][c]&&len[nx[p][c]]==len[p]+1){
379             last=nx[p][c];return;
380         }
```

```
        int np=++tot;
        memset(nx[tot],0, sizeof(nx[tot]));
        last=np; len[np]=len[p]+1;
        for(;p&&!nx[p][c];p=fa[p]) nx[p][c]=np;
        if(!p) fa[np]=1;
        else{
            int q=nx[p][c];
            if(len[p]+1==len[q]) fa[np]=q;
            else{
                int nq=++tot;
                len[nq]=len[p]+1;
                memcpy(nx[nq],nx[q], sizeof(nx[q]));
                fa[nq]=fa[q]; fa[q]=fa[np]=nq;
                for(;nx[p][c]==q;p=fa[p])
                    nx[p][c]=nq;
            }
        }
    }

    void query(){
        ll ret=0;
        for(int i=1;i<=tot;i++) c[len[i]]++;
        for(int i=1;i<=tot;i++) c[i]+=c[i-1];
        for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
        for(int i=tot;i>=1;i--){
            int p=rk[i];
            sz[0][fa[p]]+=sz[0][p]; sz[1][fa[p]]+=sz[1][p];
            ret+=(ll)sz[0][p]*sz[1][p]*(len[p]-len[fa[p]]);
        }
        printf("%lld\n",ret);
    }


}SAM;

int main(){
    SAM.init();
    scanf("%s",s);
    int len=strlen(s),p=1;
    for(int i=0;i<len;i++){
        int c=s[i]-'a';SAM.insert(c);
        p=SAM.nx[p][c];SAM.sz[0][p]++;
    }
    SAM.last=p=1;
    scanf("%s",s);
    len=strlen(s);
    for(int i=0;i<len;i++){
        int c=s[i]-'a';SAM.insert(c);
        p=SAM.nx[p][c];SAM.sz[1][p]++;
    }
    SAM.query();
}
```

2 动态规划

2.1 背包 dp

```
//m-背包容量，w-物品体积，val-物品价值，cnt-物品数量
//01背包
void OneZero(int m,int w,int val){
    for(int i=m;i>=w;i--)
        dp[i]=max(dp[i-w]+val,dp[i]);
}

//完全背包
void Com(int m,int w,int val){
    for(int i=0;i<=m-w;i++)
        dp[i+w]=max(dp[i]+val,dp[i+w]);
}

//多重背包
void Mul(int m,int w,int val,int cnt){
    if(cnt*w>=m){
        Com(m,w,val);
        return;
    }
    for(int i=1;i<=cnt;i<<=1){
        OneZero(m,w*i,val*i); cnt-=i;
    }
    if(cnt) OneZero(m,cnt*w,val*cnt);
}
```

2.2 数位 dp

```
int gcd(int a, int b) {
    return b ? gcd(b, a%b) : a;
}

ll dfs(int pos, int tot, int lcm, bool limit) {
    if (pos == 0)
        return (tot%lcm == 0);
    if (!limit&&dp[pos][ha[lcm]][tot] != -1)
        return dp[pos][ha[lcm]][tot];
    ll res = 0;
    int top = limit ? di[pos] : 9;
    for (int i = 0; i <= top; i++) {
        res += dfs(pos - 1, (tot * 10 + i) % mod, i ? i * lcm / gcd(i, lcm) : lcm, i ==
     di[pos] && limit);
    }
    if (!limit)
        dp[pos][ha[lcm]][tot] = res;
    return res;
}

ll solve(ll x) {
    int pos = 0;
    while (x) {
        di[++pos] = x % 10;
        x /= 10;
    }
    return dfs(pos, 0, 1, true);
```

```
}
```

2.3 dp 优化

```
1、斜率优化
int tail=0,head=0;
q[tail++]=0;
for(int i=1;i<=n;i++){
    //判断答案是否最优
    while(tail>1+head&&isOK(i,q[head+1],q[head])) head++;
    dp[i]=getsum(i,q[head]);
    //判断新加入的点是否最优
    while(tail>1+head&&isOK2(i,q[tail-1],q[tail-2])) tail--;
    q[tail++]=i;
}
printf("%lld\n",dp[n]);

2、四边形不等式
在dp问题中,我们常遇见这样的一类问题，他们的dp转移方程式这样的：dp[i][j]=min{dp[i][k]+dp[k+1][j]+
    cost[i][j]}

对于(a<b<=c<d)，如果有f[a][c]+f[b][d]<=f[b][c]+f[a][d]，则说明f满足四边形不等式
1、当决策代价函数w[i][j]满足w[i][j]+w[i'][j']<=w[i'][j]+w[i][j'] (i<=i'<=j<=j')时，称满足四边
    形不等式
2、当函数w[i][j]满足w[i'][j]<=w[i][j'] (i<=i'<=j<=j')时,称w关于区间包含关系单调.
结论：若决策代价函数满足四边形不等式，包含关系单调，且dp[i][j]方程也满足四边形不等式，设s[i][j]表示
dp[i][j]取得最优值时对应的下标，即(i<=k<=j)。就满足s[i][j-1]<=s[i][j]<=s[i+1][j]。

一般做法:
对于dp转移合法的证明，其实很多时候直接打表就行了，比如先跑一个O（n^3）的代码，跑的时候判断是否满足四边
    形不
等式，决策是否单增等等，如果不满足就输出false之类的，或者打一个决策表出来观察，这样其实会省下一部分时
    间。

//判断是否满足平行四边形不等式优化w[i][j]+w[i+1][j+1]<=w[i+1][j]+w[i][j+1]
bool isOK(){
    for(int i=1;i<n;i++){
        for(int j=i+2;j<n;j++){
            if(w[i][j]+w[i+1][j+1]<=w[i+1][j]+w[i][j+1]) continue;
            else return false;
        }
    }
    return true;
}


    四边形不等式优化代码十分简单,且效果也很好,但是最令人头疼的就是如何证明w满足四边形不等式。有可能这个
    对大家还比较容易，但是要
知道，满足这些性质的转移方程不止这一种！对于f[i][j] = min{ f[i-1][k] + w(k+1,j) | i-1 <= k < j
    }这个方程来说，若w满足四
边形不等式，f同样满足四边形不等式，也可以使用决策单调性优化，但是证明就比较困难了。YJQ教给我一种很好的绕
    过证明使用四边形不等式的
方法，但是使用起来不是那么简单，有一些注意事项。下面的内容可就是别人博客里没有的东西了！
    大致方法很简单，如果我们觉得一个方程能用四边形不等式优化，就把他的所有决策点，也就是p矩阵打印出来，
    观察一下在每行每列上是否单
调，如果单调，就说明这个方程可以用四边形不等式优化。不过需要小心一些地方。首先，注意决策点应该在哪些范围
    之内单调，比如对于区间dp
```

的方程来说，决策点的单调范围就应该是行号小于等于列号的那一部分。这点在实际问题中应该很容易体现出来，比如对于区间dp，行号大于列号的那些状态肯定是无用的，决策单调性也肯定不关它们什么事。

其次，应该注意递推时枚举的顺序和状态之间的依赖关系。比如对于上面那个方程来说，决策矩阵p在每行每列单调递增，所以应该把k的枚举范围该成p[i-1][j]至p[i][j+1]，注意这里和区间dp就不一样了，所以应该根据状态之间的依赖关系灵活调整枚举范围。而且，如果我们遵循这样的依赖关系，就应该有p[i][j]依赖于p[i][j+1]，所以k应该从大到小倒着枚举.

2.4 编辑距离

编辑距离:

给定两个序列S和S2,通过一系列字符编辑(插入、删除、替换)等操作，将S转变成S2，完成这种转换所需要的最少的编辑操作个数称为S和S2的编辑距离.

```
int main(){
    scanf("%s%s",s+1,s2+1);
    int len=strlen(s+1),len2=strlen(s2+1);
    dp[1][1]=s[1]==s2[1]?0:1;
    for(int i=2;i<=len;i++) dp[i][1]=dp[i-1][1]+1;
    for(int j=2;j<=len2;j++) dp[1][j]=dp[1][j-1]+1;
    for(int i=2;i<=len;i++){
        for(int j=2;j<=len2;j++){
            dp[i][j]=min(dp[i-1][j],min(dp[i][j-1],dp[i-1][j-1]))+1;
            if(s[i]==s2[j]) dp[i][j]=min(dp[i][j],dp[i-1][j-1]);
        }
    }
    for(int i=1;i<=len;i++){
        for(int j=1;j<=len2;j++){
            printf("%d ",dp[i][j]);
        }
        printf("\n");
    }
}
```

2.5 状压 dp

遍历方式:

```
//第一层从[0,pw(n)-1]的遍历方式，正好从前到后扫到所有n个位置变化状态
//第二层可以检验某一位是否存在，从而做出操作
for(int i=0;i<pw(n);i++){
    for(int j=0;j<n;j++){
        if(pw(j)&i){
            ......
        }
    }
}


//矩阵类型的遍历，通过预处理减少不必要的状态来降低复杂度
int main(){
    int x;
    scanf("%d%d",&n,&m);
    int top=pw(m);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
```

```
            scanf("%d",&x);
            g[i]=(g[i]<<1)+x; //预处理状态，将其直接保存
        }
    }
    a[0]=0;
    for(int i=0;i<top;i++){
        if((((i<<1)&i)==0)&&(((i>>1)&i)==0)){
            a[++a[0]]=i; //通过移位操作，预处理出相邻不为1的状态
        }
    }
    dp[0][0]=1;
    for(int i=1;i<=n;i++){
        //遍历矩阵上下关系来处理
        for(int j=1;j<=a[0];j++){
            ...
            for(int k=1;k<=a[0];k++){
                ...
            }
        }
    }
    ll ret=0; //答案统计
    for(int i=0;i<top;i++) ret=(ret+dp[n][i])%mod;
    printf("%lld\n",ret);
}

//初始化状态，然后用已有状态去推导接下来的状态，写起来有时候会方便很多
for(int i=0;i<n;i++) if(p[i]==-1||p[i]==0) dp[pw(i)][i]=0;
for(int i=0;i<top;i++){
    for(int j=0;j<n;j++){
        if(dp[i][j]==-inf) continue;
        for(int k=0;k<n;k++){
            if(k==j) continue;
            if(p[k]==-1||p[k]==cnt[i]){
                if(pw(k)&i) continue;
                dp[i|pw(k)][k]=max(dp[i|pw(k)][k],dp[i][j]+a[j]*a[k]);
            }
        }
    }
}


//可以快速求出所有关于S的二进制子集
int S=i;
for (int s=(S-1)&S;s;s=(s-1)&S) {
     int t=S^s;
}
```

3 数据结构

3.1 并查集

```
1、并查集模板
int find(int x){
    if (x != parent[x])
        parent[x] = find(parent[x]);
    return parent[x];
}

2、带权并查集
int find(int x){
    if (x != parent[x]){
        int t = parent[x];
        parent[x] = find(parent[x]);
        value[x] += value[t];
    }
    return parent[x];
}

void Merge(int x,int y,int v){
    int fx = find(x);
    int fy = find(y);
    if (fx != fy){
        parent[fx] = fy;
        value[fx] = -value[x] + value[y] + v;
    }
}
```

3.2 线段树

```
struct SegTree{
    ll sum[N<<2];

    static inline int lson(int k) {return k<<1;}

    static inline int rson(int k) {return k<<1|1;}

    void up(int k){
        sum[k]=sum[lson(k)]+sum[rson(k)];
    }

    void build(int k,int l,int r){
        if(l==r){
            scanf("%lld",&sum[k]);
            return;
        }
        int mid=(l+r)>>1;
        build(lson(k),l,mid);
        build(rson(k),mid+1,r);
        up(k);
    }

    void update(int k,int l,int r,int pos,int val){
        if(l==r){
            sum[k]=max(0ll,sum[k]+val);
            return;
```

```
        }
        int mid=(l+r)>>1;
        if(pos<=mid) update(lson(k),l,mid,pos,val);
        else update(rson(k),mid+1,r,pos,val);
        up(k);
    }

    ll query(int k,int l,int r,int x,int y){
        if(l==x&&r==y){
            return sum[k];
        }
        int mid = (l+r)>>1;
        if(y<=mid) return query(lson(k),l,mid,x,y);
        else if(x>mid) return query(rson(k),mid+1,r,x,y);
        else return query(lson(k),l,mid,x,mid)+query(rson(k),mid+1,r,mid+1,y);
    }
}st;
```

3.3 st 表

```
1、st表
struct ST {
    int k2[21], st[21][N], Log[N];
    void init_st(int n) {
        k2[0] = 1;
        for (int i = 1; i <= 20; i++) k2[i] = 2 * k2[i - 1];
        Log[0] = -1; for (int i = 1; i < N; i++) Log[i] = Log[i / 2] + 1;
        for (int i = 1; i <= n; i++) st[0][i] = height[i];
        for (int i = 1; i <= Log[n]; i++) {
            for (int j = 1; j + k2[i] - 1 <= n; j++) {
                st[i][j] = min(st[i - 1][j], st[i - 1][j + k2[i - 1]]);
            }
        }
    }
    int query_min(int x, int y) {
        int len = log2(y - x + 1);
        return min(st[len][x], st[len][y - k2[len] + 1]);
    }
}st;

精简版
scanf("%d%d",&n,&m);
for(int i=1;i<=n;i++)
    scanf("%d",&st[0][i]);
int top = log2(n);
for(int i=1;i<=top;i++){
    for(int j=1;j+(1<<i)-1<=n;j++){
        st[i][j]=min(st[i-1][j],st[i-1][j+(1<<i-1)]);
    }
}
int x=1,y=m;
while(y<=n){
     int len = log2(y-x+1);
     printf("%d\n",min(st[len][x],st[len][y-(1<<len)+1]));
     x++;y++;
}
```

3.4 线性基

```
struct Linear_Basis{
    ll d[61],p[61];
    int cnt;
    bool zero;
    //初始化
    Linear_Basis(){
        memset(d,0,sizeof(d));
        memset(p,0,sizeof(p));
        cnt=0; zero=0;
    }
    //插入
    bool insert(ll val){
        for (int i=60;i>=0;i--)
            if (val&(1LL<<i)){
                if (!d[i]){
                    d[i]=val;
                    break;
                }
                val^=d[i];
            }
        return val>0;
    }
    //查询线性基所能表示最大值
    ll query_max(){
        ll ret=0;
        for (int i=60;i>=0;i--)
            if ((ret^d[i])>ret)
                ret^=d[i];
        return ret;
    }
    //查询线性基所能表示最小值
    ll query_min(){
        for (int i=0;i<=60;i++)
            if (d[i])
                return d[i];
        return 0;
    }
    //重构,消除多余的1
    void rebuild(){
        for (int i=60;i>=0;i--)
            for (int j=i-1;j>=0;j--)
                if (d[i]&(1LL<<j))
                    d[i]^=d[j];
        for (int i=0;i<=60;i++)
            if (d[i])
                p[cnt++]=d[i];
    }
    //求第k小数,注意特判0
    ll kthquery(ll k){
        ll ret=0;
        if (k>=(1LL<<cnt))
            return -1;
        for (int i=60;i>=0;i--)
            if (k&(1LL<<i))
                ret^=p[i];
        return ret;
    }
```

```
    //查询是否存在x->y的变化
    bool query_exist(ll x,ll y){
        for(int i=60;i>=0;i--){
            if((1ll<<i)&y){
                if((1ll<<i)&x)
                    continue;
                else
                    x^=d[i];
            }else if((1ll<<i)&x){
                x^=d[i];
            }
        }
        return x==y;
    }
};

//线性基合并
Linear_Basis merge(const Linear_Basis &n1,const Linear_Basis &n2){
    Linear_Basis ret=n1;
    for (int i=60;i>=0;i--)
        if (n2.d[i])
            ret.insert(n1.d[i]);
    return ret;
}
```

3.5 树分治

```
//多组测试数据，每次输入n、m，和一棵n个点的有边权的树，问你满足x到y距离小于等于m的无序点对(x,y)的个数是多少。
int n,k,head[N],tot;
bool vis[N];
int part[N],cnt[N],root,deep[N],d[N],dtot,sn;
ll ans;
struct node{
    int to,nx,val;
}edge[N*2];

void add_edge(int from,int to,int val){
    edge[tot].to=to;
    edge[tot].val=val;
    edge[tot].nx=head[from];
    head[from]=tot++;
}

//求取重心
void getRoot(int from,int pre){
    cnt[from]=1; part[from]=0;
    for(int i=head[from];i;i=edge[i].nx){
        int to=edge[i].to;
        if(to==pre||vis[to]) continue;
        getRoot(to,from); cnt[from]+=cnt[to];part[from]=max(part[from],cnt[to]);
    }
    part[from]=max(part[from],sn-cnt[from]);
    if(part[root]>part[from]) root=from;
}

//求取树深度
void getDeep(int from,int pre){
```

```
    d[++dtot]=deep[from];
    for(int i=head[from];i;i=edge[i].nx){
        int to=edge[i].to;
        if(to==pre||vis[to]) continue;
        deep[to]=deep[from]+edge[i].val;
        getDeep(to,from);
    }
}

//计算答案贡献
ll calc(int from){
    dtot=0; getDeep(from,0);
    sort(d+1,d+1+dtot);
    ll sum=0;
    int i=1,j=dtot;
    while(i<j){
        if(d[i]+d[j]<=k) sum+=j-i,i++;
        else j--;
    }
    return sum;
}

//分治
void Divide(int from){
    deep[from]=0; ans+=calc(from); vis[from]=1;
    for(int i=head[from];i;i=edge[i].nx){
        int to=edge[i].to;
        if(vis[to]) continue;
        deep[to]=edge[i].val; ans-=calc(to); sn=part[to];
        root=0; getRoot(to,0); Divide(root);
    }
}

int main() {
    part[0]=inf;
    int u,v,w;
    while(scanf("%d%d",&n,&k)==2){
        if(n==0&&k==0) break;
        tot=1;
        memset(head,0, sizeof(int)*(n+1));
        memset(vis,0, sizeof(bool)*(n+1));
        for(int i=1;i<n;i++){
            scanf("%d%d%d",&u,&v,&w);
            add_edge(u,v,w); add_edge(v,u,w);
        }
        ans=root=0; sn=n;
        getRoot(1,0); Divide(root);
        printf("%lld\n",ans);
    }
    return 0;
}
```

3.6 树重心

```
int n;
vector<int>e[N];
int min_p,min_part;
int node[N];
```

```
//树的重心也叫树的质心。对于一棵树n个节点的无根树，找到一个点，使得把树变成以该点为根的有根
//树时，最大子树的结点数最小。换句话说，删除这个点后最大连通块（一定是树）的结点数最小。
//性质:
//1、树中所有点到某个点的距离和中，到重心的距离和是最小的，如果有两个距离和，他们的距离和一样。
//2、把两棵树通过一条边相连，新的树的重心在原来两棵树重心的连线上。
//3、一棵树添加或者删除一个节点，树的重心最多只移动一条边的位置。
//4、一棵树最多有两个重心，且相邻。

//min_p,min_part是求得重心

void dfs(int from,int pre){
    node[from]=1; int max_part=0;
    for(int i=0;i<e[from].size();i++){
        int to=e[from][i];
        if(pre==to) continue;
        dfs(to,from);
        node[from]+=node[to];
        max_part=max(max_part,node[to]);
    }
    max_part=max(max_part,n-node[from]);
    if(max_part<min_part){
        min_p=from; min_part=max_part;
    }
}

int main() {
    int t,u,v;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        for(int i=1;i<=n;i++) e[i].clear();
        for(int i=1;i<n;i++){
            scanf("%d%d",&u,&v);
            e[u].push_back(v); e[v].push_back(u);
        }
        min_p=0; min_part=inf;
        dfs(1,0);
        printf("%d %d\n",min_p,min_part);
    }
    return 0;
}
```

3.7 树状数组

```
树状数组:
    树状数组常用于维护前缀信息，如前缀和、前缀乘积等等。最常见的如给你一个长度为n的数组，单点修改，查询区间的和。
    在树状数组中，第i个点储存了右端点为i，区间长度为lowbit(i)的区间元素和，也就是从i-lowbit(i)+1到i的区间和
Lowbit(x)表示求x在二进制下位最低的1连同后面的0所组成的数字，举个例子，6的二进制表示是110(2)，那么lowbit(6)=10(2)=2，也就是lowbit(6)=2

//注意0
//对一个数组进行区间修改可以使用差分思想，即初始是c[N]={0},修改时add(l,val),add(r+1,-val),getsum(i)+a[i]即为更新后结果
struct BIT{
    ll c[N];

```

```
    inline int lowbit(int x) {return x&-x;}

    void add(int x,int val){
        for(;x<=n;x+=lowbit(x)){
            c[x]+=val;
        }
    }

    ll sum(int x){
        ll sum=0;
        for(;x>0;x-=lowbit(x)){
            sum+=c[x];
        }
        return sum;
    }
};
```

树状数组维护前缀和的前缀和:
主要应用在以下两个方面: 1、区间加数，区间求和问题。2、区间加等差数列，单点求值问题。
设s1表示数组a的前缀和，s2表示s1的前缀和，则有:
s1[i]=sigma(j=1,i)a[j]
s2[i]=sigma(j=1,i)s1[j]
经化简可得: s2[i]=(i+1)sigma(j=1,i)a[j]-sigma(j=1,i)j*a[j]
因此我们只需要开两个数组维护(i+1)sigma(j=1,i)a[j]和sigma(j=1,i)j*a[j]即可。

更高阶情况查看纸质模板

3.8 树的直径

树的直径:
概念:
树上距离最远的点（树的最长路）。
思想
在树上任选一点u，求距离点u最远的点v，再求距离点v最远的点s，点v到点s的距离即为树的直径。
过程:
两遍BFS :先任选一个起点BFS找到最长路的终点，再从终点进行BFS，则第二次BFS找到的最长路即为树的直径； 原理: 设起点为u,第一次BFS找到的终点v一定是树的直径的一个端点。
证明:
1）如果u是直径上的点，则v显然是直径的终点(因为如果v不是的话，则必定存在另一个点w使得u到w的距离更长，则于BFS找到了v矛盾)
2）如果u不是直径上的点，则u到v必然于树的直径相交(反证),那么交点到v必然就是直径的后半段了所以v一定是直径的一个端点，所以从v进行BFS得到的一定是直径长度

4 图论

4.1 最短路

```
1、Dijkstra
//不能处理负权图，复杂度(V+E)logV
int n,m,tot,head[N];
ll dis[N];
bool vis[N];
struct node{
    int id;
    ll d;
    node(){}
    node(int id,ll d):id(id),d(d){}
    bool operator < (const node& x) const {
        return d>x.d;
    }
};

struct edge{
    int to,nx;
    ll w;
}e[N<<1];

//tot初始化为1
void add_edge(int from,int to,ll w){
    e[tot].to=to; e[tot].w=w;
    e[tot].nx=head[from]; head[from]=tot++;
}

void Dijkstra(int st){
    memset(dis,0x3f, sizeof(ll)*(n+1));
    memset(vis,0, sizeof(bool)*(n+1));
    dis[st]=0;
    priority_queue<node>q;
    node y(st,0);
    q.push(y);
    while(!q.empty()){
        node x=q.top(); q.pop();
        if(vis[x.id]) continue;
        vis[x.id]=1;
        for(int i=head[x.id];i;i=e[i].nx){
            int to=e[i].to,w=e[i].w;
            if(vis[to]) continue;
            if(w+dis[x.id]<dis[to]){
                dis[to]=w+dis[x.id];
                y.d=dis[to]; y.id=to;
                q.push(y);
            }
        }
    }
}

2、Floyd
//可以求任意两个点的最短路，以及输出字典序的路径
int n;
int link[N][N],path[N][N];
int w[N];

```

```
void Floyd(int n){
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                if(link[i][k]+link[k][j]+w[k]<link[i][j]){
                    link[i][j]=link[i][k]+link[k][j]+w[k];
                    path[i][j]=path[i][k];
                }else if(link[i][k]+link[k][j]+w[k]==link[i][j]&&path[i][j]>path[i][k])
    {
                    path[i][j]=path[i][k];
                }
            }
        }
    }
}

int main() {
    int x;
    while(scanf("%d",&n)==1){
        if(n==0) break;
        for(int i=1;i<=n;i++) for(int j=1;j<=n;j++){
            scanf("%d",&x);
            if(x==-1) link[i][j]=inf;
            else link[i][j]=x;
            path[i][j]=j;
        }
        for(int i=1;i<=n;i++) scanf("%d",&w[i]);
        Floyd(n);
        int start,endd;
        while(scanf("%d%d",&start,&endd)==2){
            if(start==endd&&start==-1) break;
            printf("From %d to %d :\nPath: ",start,endd);
            int x=start;
            printf("%d",x);
            while(x!=endd){
                printf("-->%d",path[x][endd]);
                x=path[x][endd];
            }
            printf("\nTotal cost : %d\n\n",link[start][endd]);
        }

    }
}

题意:
    有N个城市，然后直接给出这些城市之间的邻接矩阵，矩阵中-1代表那两个城市无道路相连，其他值代表路径长
    度。
如果一辆汽车经过某个城市，必须要交一定的钱。现在要从a城到b城，花费为路径长度之和，再加上除起点与终点外所
有城市的过路费之和。求最小花费，如果有多条路经符合，则输出字典序最小的路径。

//输入
5
0 3 22 -1 4
3 0 5 -1 -1
22 5 0 9 20
-1 -1 9 0 4
4 -1 20 4 0
5 17 8 3 1
1 3
```

```
3 5
2 4
-1 -1
0

//输出
From 1 to 3 :
Path: 1-->5-->4-->3
Total cost : 21

From 3 to 5 :
Path: 3-->4-->5
Total cost : 16

From 2 to 4 :
Path: 2-->1-->5-->4
Total cost : 17

3、spfa
//主要拿来判断负环，负环的概念为环中所有权值和为负数，只需要有个点重复使用次数>n，那么存在负环
bool spfa(int st){
    memset(dis,0x3f, sizeof(ll)*(n+1));
    memset(vis,0, sizeof(bool)*(n+1));
    memset(cnt,0, sizeof(int)*(n+1));
    queue<int>q;
    vis[st]=1;dis[st]=0;q.push(st);cnt[st]++;
    while(!q.empty()){
        int from=q.front();
        q.pop(); vis[from]=0;
        for(int i=head[from];i;i=edge[i].nx){
            int to=edge[i].to;
            if(dis[to]>dis[from]+edge[i].w){
                dis[to]=dis[from]+edge[i].w;
                if(!vis[to]){
                    vis[to]=1;q.push(to); cnt[to]=cnt[from]+1;
                    if(cnt[to]>n) return true;
                }
            }
        }
    }
    return false;
}

//Floyd判负环
bool Floyd(int n){
    for(int k=1;k<=n;k++){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
                int t=link[i][k]+link[k][j];
                if(link[i][j]>t)link[i][j]=t;
            }
            if(link[i][i]<0) return true;
        }
    }
    return false;
}
```

4.2 最小生成树

把一个连通无向图的生成树边按权值递增排序，称排好序的边权列表为有序边权列表，则任意两棵最小生成树的有序边权列表是相同的。

1、Prim

```
//时间复杂度(V+E)logV

struct node{
    int id,w;
    bool operator <(const node&x) const {
        return w>x.w;
    }
};

ll Prim(){
    ll res=0; node x; int cnt=0;
    priority_queue<node>q; q.push(node{1,0});
    while(!q.empty()){
        node y=q.top(); q.pop();
        if(vis[y.id]) continue;
        res+=y.w; vis[y.id]=1; cnt++;
        for(int i=head[y.id];i;i=edge[i].nx){
            int to=edge[i].to,w=edge[i].w;
            if(vis[to]) continue;
            q.push(node{to,w});
        }
    }
    if(cnt==n) return res;
    return -1;
}
```

2、Kruscal

```
//时间复杂度ElogE
//将权值从大到小排列可生成最大生成树

int n,m,tot,parent[N];
struct node{
    int from,to,w;
}edge[N];

int find(int x){
    if (x != parent[x]) parent[x] = find(parent[x]);
    return parent[x];
}

bool cmp(node x,node y){
    return x.w<y.w;
}

ll Kruskal(){
    ll res=0;
    for(int i=1;i<=n;i++) parent[i]=i;
    sort(edge+1,edge+1+m,cmp); int cnt=0;
    for(int i=1;i<=m;i++){
        if(cnt==n-1) return res;
        int fx=find(edge[i].from),fy=find(edge[i].to);
        if(fx!=fy){
            parent[fx]=fy;
```

```
            res+=edge[i].w;
            cnt++;
        }
    }
    if(cnt!=n-1) return -1;
    else return res;
}

//检验最小生成树唯一性可以使用Kruscal求出所有边，然后将逐一枚举将边去掉，尝试是否能生成最小生成树
ll Kruskal(){
    ll ret=0; int cnt=0; v[0]=0;
    for(int i=1;i<=n;i++) parent[i]=i;
    sort(edge+1,edge+1+m,cmp);
    for(int i=1;i<=m;i++){
        if(cnt==n-1) break;
        int fx=find(edge[i].from),fy=find(edge[i].to);
        if(fx!=fy){
            parent[fx]=fy; cnt++; ret+=edge[i].w; v[++v[0]]=i;
        }
    }
    for(int i=1;i<=v[0];i++){
        ll sum=0; cnt=0;
        for(int j=1;j<=n;j++) parent[j]=j;
        for(int j=1;j<=m;j++){
            if(j==v[i]) continue;
            int fx=find(edge[j].from),fy=find(edge[j].to);
            if(fx!=fy){
                parent[fx]=fy; cnt++; sum+=edge[j].w;
            }
            if(cnt==n-1&&ret==sum) return -1;
            else if(cnt==n-1) break;
        }
    }
    return ret;
}
```

4.3 强联通分量

```
1、tarjan
//初始化dfs=top=cnt=0,tot=1;
//如果多组数据head[N], dfn[N], low[N], Stack[N], color[N]都得初始化
//用来判强联通分量，若存在路径u->v,v->u，那么会归并到一个集合之中
//时间复杂度为O(v+e)
//若将一个图n个点转化为一整个强联通分量，只需要统计整个图出度和入度，答案为其中最大值
int n, m;
int head[N], dfn[N], low[N], Stack[N], color[N];
bool vis[N];
int dfs, tot, top, cnt;

struct node {
    int from, to, nx;
} edge[M];

void add_edge(int from, int to) {
    edge[tot].from = from;
    edge[tot].to = to;
    edge[tot].nx = head[from];
    head[from] = tot++;
```

```
}

void Tarjan(int x) {
    dfn[x] = ++dfs; low[x] = dfs;
    vis[x] = 1; Stack[++top] = x;
    for (int i = head[x]; i; i = edge[i].nx) {
        int tmp = edge[i].to;
        if (!dfn[tmp]) {
            Tarjan(tmp); low[x] = min(low[tmp], low[x]);
        } else if (vis[tmp]) low[x] = min(low[x], dfn[tmp]);
    }
    if (dfn[x] == low[x]) {
        vis[x] = 0; color[x] = ++cnt;
        while (Stack[top] != x) {
            color[Stack[top]] = cnt;
            vis[Stack[top--]] = false;
        }
        top--;
    }
}

void gao(){
    //多组注意初始化
    for(int i=1;i<=n;i++){
        if(color[i]==0) Tarjan(i);
    }
    for(int i=1;i<=n;i++) printf("%d ",color[i]);
    //重新建图，进行操作
}

2、kosaraju
//通过两次dfs得到强联通分量，注意要正反建图，初始化tot=tot2=1
//时间复杂度为O(v+e)

int n,m,cnt;
int head[N],head2[N],tot,tot2;
int color[N];
bool vis[N];
struct node{
    int to,nx;
}edge[N],edge2[N];
stack<int>st;

void dfs1(int from){
    vis[from]=1;
    for(int i=head[from];i;i=edge[i].nx){
        int to=edge[i].to;
        if(!vis[to]) dfs1(to);
    }
    st.push(from);
}

void dfs2(int from){
    color[from]=cnt;
    for(int i=head2[from];i;i=edge2[i].nx){
        int to=edge2[i].to;
        if(!color[to]) dfs2(to);
    }
}
```

```

void Kosaraju(){
    memset(color,0, sizeof(int)*(n+1));
    memset(vis,0, sizeof(bool)*(n+1)); cnt=0;
    for(int i=1;i<=n;i++)
        if(!vis[i]) dfs1(i);
    while(!st.empty()){
        int x=st.top(); st.pop();
        if(!color[x]){
            cnt++;
            dfs2(x);
        }
    }
}
```

4.4 网络流

```
1、dinic
struct Edge {
    int e, nxt;
    ll v;

    Edge() = default;

    Edge(int a, ll b, int c = 0) : e(a), v(b), nxt(c) {}

    bool operator<(const Edge& a) const {
        return (a.v == v ? e < a.e : v < a.v);
    }

    bool operator>(const Edge& a) const {
        return (a.v == v ? e > a.e : v > a.v);
    }
};

struct Graph {
    Edge eg[M];
    int head[N];
    int cnt;

    void init(int n) {
        memset(head, -1, sizeof(int) * ++n);
        cnt = 0;
    }

    inline void addEdge(int x, int y, ll v) {
        eg[cnt] = Edge(y, v, head[x]);
        head[x] = cnt++;
    }
} gh;


struct Dinic {
    Graph gh;
    // 点的范围[0, n)
    int n;
    // 弧优化
    int cur[N], dis[N];
```

```
42
43      Dinic() {};
44
45      // 设置N
46      void init(int _n) {
47          n = _n + 1;
48          gh.init(n);
49      }
50
51      // 加流量
52      void addFlow(int x, int y, ll f) {
53          gh.addEdge(x, y, f);
54          gh.addEdge(y, x, 0);
55      }
56
57      bool bfs(int s, int e) {
58          memset(dis, -1, sizeof(int) * n);
59          int q[N];
60          int l, r;
61          l = r = 0;
62          dis[s] = 0;
63          q[r++] = s;
64          while (l < r) {
65              int f = q[l++];
66              for (int i = gh.head[f]; ~i; i = gh.eg[i].nxt) {
67                  if (gh.eg[i].v > 0 && dis[gh.eg[i].e] == -1) {
68                      dis[gh.eg[i].e] = dis[f] + 1;
69                      q[r++] = gh.eg[i].e;
70                  }
71              }
72          }
73          return dis[e] > 0;
74      }
75
76      ll dfs(int s, int e, ll mx) {
77          if (s == e || mx == 0) {
78              return mx;
79          }
80          ll flow = 0;
81          for (int& k = cur[s]; ~k; k = gh.eg[k].nxt) {
82              auto& eg = gh.eg[k];
83              ll a;
84              if (eg.v > 0 && dis[eg.e] == dis[s] + 1 && (a = dfs(eg.e, e, min(eg.v, mx))
    )) {
85                  eg.v -= a;
86                  gh.eg[k ^ 1].v += a;
87                  flow += a;
88                  mx -= a;
89                  if (mx <= 0) break;
90              }
91          }
92          return flow;
93      }
94
95      ll max_flow(int s, int e) {
96          ll ans = 0;
97          while (bfs(s, e)) {
98              memcpy(cur, gh.head, sizeof(int) * n);
99              ans += dfs(s, e, INF);
```

```
        }
        return ans;
    }
} dinic;
```

4.5 最小树形图

```
朱刘算法(O(VE))
一、相关定义
定义：设G = (V,E)是一个有向图，它具有下述性质：1、G中不包含有向环；2、存在一个顶点vi，它不是任何弧的终点，而V中的
其它顶点都恰好是唯一的一条弧的终点，则称 G是以vi为根的树形图。
最小树形图就是有向图G = (V, E)中以vi为根的树形图中权值和最小的那一个。
另一种说法：最小树形图，就是给有向带权图一个特殊的点root，求一棵以root为根节点的树使得该树的的总权值最小。
性质：最小树形图基于贪心和缩点的思想。
缩点：将几个点看成一个点，所有连到这几个点的边都视为连到收缩点,所有从这几个点连出的边都视为从收缩点连出

算法概述：
为了求一个图的最小树形图，1、先求出最短弧集合E0；2、如果E0不存在，则图的最小树形图也不存在；3、如果E0存在且不具有环，
则E0就是最小树形图；4、如果E0存在但是存在有向环，则把这个环收缩成一个点u，形成新的图G1，然后对G1继续求其的最小树形图，
直到求到图Gi，如果Gi不具有最小树形图，那么此图不存在最小树形图，如果Gi存在最小树形图，那么逐层展开，就得到了原图的最
小树形图。

模板1:
题意：
    给出n(1<=n<=1000)个点,m(1<=m<=10000)条边，求出最小树形图，并输出根节点，点从0开始
输入：
3 1
0 1 1

4 4
0 1 10
0 2 10
1 3 20
2 3 30

输出：
impossible

40 0

int n,m,pos,pre[N],id[N],vis[N];
//in[i]存最小入边权,pre[v]为该边的起点
ll in[N];
struct node{
    int u,v;
    ll w;
}edge[M];

ll Directed_MST(int root,int V,int E){
    //存最小树形图总权值
    ll ret=0;
    while(1){
        //1.找每个节点的最小入边
```

```
47          for(int i=0;i<V;i++) in[i]=INF;
48          for(int i=0;i<E;i++){
49              int u=edge[i].u,v=edge[i].v;
50              if(edge[i].w<in[v]&&u!=v){
51                  in[v]=edge[i].w;
52                  pre[v]=u;
53                  //这个点就是实际的起点
54                  if(root==u) pos=i;
55              }
56          }
57          //判断是否存在最小树形图
58          for(int i=0;i<V;i++){
59              if(i==root) continue;
60              //除了根以外有点没有入边,则根无法到达它说明它是独立的点 一定不能构成树形图
61              if(in[i]==INF) return -1;
62          }
63          //2.找环
64          int cnt=0;
65          memset(id,-1, sizeof(int)*(n+1));
66          memset(vis,-1, sizeof(int)*(n+1));
67          in[root]=0;
68          for(int i=0;i<V;i++){
69              ret+=in[i];
70              int v=i;
71              while(vis[v]!=i&&id[v]==-1&&v!=root){
72                  vis[v]=i;
73                  v=pre[v];
74              }
75              if(v!=root&&id[v]==-1){
76                  for(int u=pre[v];u!=v;u=pre[u]) id[u]=cnt;
77                  id[v]=cnt++;
78              }
79          }
80          if(cnt==0) break; //无环则break
81          for(int i=0;i<V;i++){
82              if(id[i]==-1) id[i]=cnt++;
83          }
84          ////3.建立新图   缩点,重新标记
85          for(int i=0;i<E;i++){
86              int u=edge[i].u,v=edge[i].v;
87              edge[i].u=id[u]; edge[i].v=id[v];
88              if(id[u]!=id[v]){
89                  edge[i].w-=in[v];
90              }
91          }
92          V=cnt;
93          root=id[root];
94      }
95      return ret;
96  }
97
98  int main() {
99      while(scanf("%d%d",&n,&m)==2){
100         ll sum=0;
101         for(int i=0;i<m;i++){
102             scanf("%d%d%lld",&edge[i].u,&edge[i].v,&edge[i].w);
103             edge[i].u++;edge[i].v++;
104             sum+=edge[i].w;
105         }
```

```
        sum++;
        //增加超级节点0,节点0到其余各个节点的边权相同（此题中边权要大于原图的总边权值）
        for(int i=m;i<n+m;i++){
            edge[i].u=0; edge[i].v=i-m+1; edge[i].w=sum;
        }
        ll ans=Directed_MST(0,n+1,m+n);
        //n+1为总结点数,m+n为总边数
        //ans代表以超级节点0为根的最小树形图的总权值,
        //将ans减去sum,如果差值小于sum,说明节点0的出度只有1,说明原图是连通图
        //如果差值>=sum,那么说明节点0的出度不止为1,说明原图不是连通图
        if(ans==-1||ans-sum>=sum) printf("impossible\n\n");
        else printf("%lld %d\n\n",ans-sum,pos-m);
    }
    return 0;
}
```

模板2:
题意:
    给定包含n个结点，m条有向边的一个图。试求一棵以结点r为根的最小树形图，并输出最小树形图每条边的权值之和，如果没有以r为根的最小树形图，输出 -1。

输入:

```
4 6 1 //n,m,root
1 2 3
1 3 1
4 1 2
4 2 2
3 2 1
3 4 1
```

输出:

```
3
```

```
int n,m,pos,pre[N],id[N],vis[N],root;
//in[i]存最小入边权,pre[v]为该边的起点
ll in[N];
struct node{
    int u,v,w;
}edge[M];

ll Directed_MST(int root,int V,int E){
    ll ret=0;
    while(1){
        for(int i=1;i<=V;i++) in[i]=INF;
        for(int i=1;i<=E;i++){
            int u=edge[i].u,v=edge[i].v;
            if(edge[i].w<in[v]&&u!=v){
                in[v]=edge[i].w;
                pre[v]=u;
            }
        }
        for(int i=1;i<=V;i++){
            if(root==i) continue;
            if(in[i]==INF) return -1;
        }
        int cnt=0;
        memset(id,-1, sizeof(int)*(n+1));
        memset(vis,-1, sizeof(int)*(n+1));
        in[root]=0;
```

```
        for(int i=1;i<=V;i++){
            ret+=in[i];
            int v=i;
            while(vis[v]!=i&&id[v]==-1&&v!=root){
                vis[v]=i;
                v=pre[v];
            }
            if(v!=root&&id[v]==-1){
                id[v]=++cnt;
                for(int u=pre[v];u!=v;u=pre[u]) id[u]=cnt;
            }
        }
        if(cnt==0) break; //无环则break
        for(int i=1;i<=V;i++){
            if(id[i]==-1) id[i]=++cnt;
        }
        ////3.建立新图,缩点,重新标记
        for(int i=1;i<=E;i++){
            int u=edge[i].u,v=edge[i].v;
            edge[i].u=id[u]; edge[i].v=id[v];
            if(id[u]!=id[v]){
                edge[i].w-=in[v];
            }
        }
        V=cnt;
        root=id[root];
    }
    return ret;
}

int main(){
    scanf("%d%d%d",&n,&m,&root);
    for(int i=1;i<=m;i++) scanf("%d%d%d",&edge[i].u,&edge[i].v,&edge[i].w);
    printf("%lld\n",Directed_MST(root,n,m));
    return 0;
}
```

4.6 割点、桥、双联通分量

```
1、割点
在无向连通图中，如果将其中一个点以及所有连接该点的边去掉，图就不再连通，那么这个点就叫做割点

Tarjan算法
    可以使用Tarjan算法求割点（注意，还有一个求连通分量的算法也叫Tarjan算法，与此算法类似）。首先选定一个根节点，从该根节
点开始遍历整个图（使用DFS）。对于根节点，判断是不是割点很简单——计算其子树数量，如果有2棵即以上的子树，就是割点。因为如果
去掉这个点，这两棵子树就不能互相到达。对于非根节点，判断是不是割点就有些麻烦了。我们维护两个数组dfn[]和low[]，dfn[u]表示
顶点u第几个被（首次）访问，low[u]表示顶点u及其子树中的点，通过非父子边（回边），能够回溯到的最早的点（dfn最小）的dfn值（
但不能通过连接u与其父节点的边）。对于边(u，v)，如果low[v]>=dfn[u]，此时u就是割点。但这里也出现一个问题：怎么计算low[u]。
假设当前顶点为u，则默认low[u]=dfn[u]，即最早只能回溯到自身。有一条边(u，v)，如果v未访问过，继续DFS，DFS完之后，low[u]=
min(low[u]，low[v])；如果v访问过（且u不是v的父亲），就不需要继续DFS了，一定有dfn[v]<dfn[u]，low[u]=min(low[u]，dfn[v])。

```

代码:

```
//dfs初始为0,tot=0, head=-1
int n,m,head[N],tot,dfn[N],low[N],dfs,iscut[N];
struct node{
    int v,nx;
}edge[M<<1];

void add_edge(int u,int v){
    edge[tot].v=v;
    edge[tot].nx=head[u];head[u]=tot++;
}

//求出所有割点,id为上一条边
void Tarjan(int u,int id){
    int cnt=0;
    dfn[u]=low[u]=++dfs;
    for(int i=head[u];~i;i=edge[i].nx){
        int v=edge[i].v;
        if(i==(id^1)) continue;
        if(!dfn[v]){
            cnt++; Tarjan(v,i);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]) iscut[u]=1;
        }else low[u]=min(low[u],dfn[v]);
    }
    if(cnt==1&&id==-1) iscut[u]=0;
}


//求一个图中去掉两个点后的最大连通块数
int n,m,head[N],tot,dfn[N],low[N],dfs,iscut[N];
struct node{
    int v,nx;
}edge[M];

void add_edge(int u,int v){
    edge[tot].v=v;
    edge[tot].nx=head[u];
    head[u]=tot++;
}

//一个点时父节点iscut=0, iscut>0时就是砍掉该点后连通块个数
void Tarjan(int u,int pre,int ban){
    int cnt=0;
    dfn[u]=low[u]=++dfs;
    for(int i=head[u];i;i=edge[i].nx){
        int v=edge[i].v;
        if(v==pre||v==ban) continue;
        if(!dfn[v]){
            cnt++; Tarjan(v,u,ban);
            low[u]=min(low[u],low[v]);
            if(low[v]>=dfn[u]) iscut[u]++;
        }else low[u]=min(low[u],dfn[v]);
    }
    //if(cnt==1&&pre==-1) iscut[u]=0;
}

int main() {
```

```
    int x,y;
    while(scanf("%d%d",&n,&m)==2){
        memset(head,0, sizeof(int)*(n+1)); tot=1;
        for(int i=1;i<=m;i++){
            scanf("%d%d",&x,&y);
            add_edge(x,y); add_edge(y,x);
        }
        int ret=0;
        //去掉一个点后，求割点
        for(int j=0;j<n;j++){
            memset(dfn,0, sizeof(int)*(n+1));dfs=0;
            //对节点进行初始化
            for(int i=0;i<n;i++) iscut[i]=1; iscut[j]=0;
            int k=0;
            for(int i=0;i<n;i++){
                if(i==j||dfn[i]) continue;
                iscut[i]=0; k++;
                Tarjan(i,i,j);
            }
            for(int i=0;i<n;i++){
                if(i==j) continue;
                ret=max(ret,k+iscut[i]-1);
            }
        }
        printf("%d\n",ret);
    }
}
```

2、桥

对于一个无向图，如果删掉一条边后图中的连通分量数增加了，则称这条边为桥或者割边。

和割点差不多，只要改一处：low[v]>dfn[u]就可以了，而且不需要考虑根节点的问题。

割边是和是不是根节点没关系的，原来我们求割点的时候是指点v是不可能不经过父节点为回到祖先节点（包括父节点），所以顶点u是割点。

如果low[v]==dfn[u]表示还可以回到父节点，如果顶点v不能回到祖先也没有另外一条回到父亲的路，那么u-v这条边就是割边。

```
//如果图中有重边，且允许两个点形成一个环，则需修改对能否访问父节点的判断，即若当前边指向父节点，但不是从父节点走到当前点的边，
//则可以用父节点的dfn更新当前点的low。
//桥上有防卫，现在要去炸一条桥，使得图不连通，求最少需要的士兵数。
//如果桥上没有防卫也需要一个士兵去炸桥，如果图不连通那么不需要士兵，重边存在则不可能是该边被炸

//dfs初始为0,tot=0，head=-1
int n,m,head[N],tot,dfn[N],low[N],dfs;
int ret;
struct node{
    int v,w,nx;
}edge[M<<1];

void add_edge(int u,int v,int w){
    edge[tot].v=v;edge[tot].w=w;
    edge[tot].nx=head[u];head[u]=tot++;
}

//求出所有割点,id为上一条边
void Tarjan(int u,int id){
    dfn[u]=low[u]=++dfs;
    for(int i=head[u];~i;i=edge[i].nx){
        int v=edge[i].v;
```

```
        if(i==(id^1)) continue;
        if(!dfn[v]){
            Tarjan(v,i);
            low[u]=min(low[u],low[v]);
            //或low[v]>dfn[u]
            if(low[v]==dfn[v]) {
                //edge[i]为桥
                ret=min(ret,edge[i].w);
            }
        }else low[u]=min(low[u],dfn[v]);
    }
}

int main() {
    int u,v,w;
    while(scanf("%d%d",&n,&m)==2){
        if(n==0&&m==0) break;
        tot=0;memset(head,-1, sizeof(int)*(n+1));
        for(int i=1;i<=m;i++){
            scanf("%d%d%d",&u,&v,&w);
            add_edge(u,v,w); add_edge(v,u,w);
        }
        int cnt=0;ret=inf;
        memset(dfn,0, sizeof(int)*(n+1)); dfs=0;
        for(int i=1;i<=n;i++){
            if(!dfn[i]){
                cnt++; Tarjan(i,-1);
            }
        }
        if(cnt>1){
            printf("0\n");
            continue;
        }
        if(ret==inf){
            printf("-1\n");
            continue;
        }
        printf("%d\n",ret==0?1:ret);
    }
}


3、双联通分量
在一张连通的无向图中，对于两个点u和v，如果无论删去哪条边（只能删去一条）都不能使它们不连通，我们就说u和v边双连通 。
在一张连通的无向图中，对于两个点u和v，如果无论删去哪个点（只能删去一个，且不能删u和v自己）都不能使它们不连通，我们就说u和点双连通 。
边双连通具有传递性，即，若x,y边双连通，y,z边双连通，则x,z边双连通。点双连通不具有传递性.
求解点双连通分量与边双连通分量其实和求解割点与桥密切相关。不同双连通分量最多只有一个公共点，即某一个割点，任意一个割点都是至少两个
点双连通的公共点。不同边双连通分量没有公共点，而桥不在任何一个边双连通分量中，点双连通分量一定是一个边双连通分量。

怎么判断一个双连通分量中环的个数呢？根据点数跟边数的关系
1.当点数=边数，形成一个环
2.当点数>边数（一条线段，说明这条边是桥）
3.当点数<边数，那么就含1个以上的环了

int co,color[N];
```

```
183 //co初始化为0,颜色相同则说明是属于同一个双联通分量之中，注意根节点为0,若其他点也为0，则说明改点也属于根
        节点双联通分量中
184
185 //边双联通分量
186 //如何将一个图补成边双联通分量，将图中已有双联通分量合并，然后形成一棵树，统计所以度为1的节点，那么最小
        值为(leaf+1)/2
187
188
189 void Tarjan(int u,int id,int cnt){
190     low[u]=dfn[u]=++dfs;
191     bcc[u]=cnt; st.push(u);
192     for(int i=head[u];~i;i=edge[i].nx){
193         int v=edge[i].v;
194         if(id==(i^1)) continue;
195         if(!dfn[v]) {
196             Tarjan(v, i,cnt);
197             low[u] = min(low[u],low[v]);
198         }else low[u]=min(low[u],low[v]);
199     }
200     if(dfn[u]==low[u]){
201         blocks++;
202         int curr;
203         do{
204             curr=st.top();
205             st.pop();
206             ebc[curr]=blocks;
207         }while(curr!=u);
208     }
209 }
210
211
212 //一个无向连通图中，每个点都有值，现在求去掉一个桥后，得到的两个连通图价值和之差最小
213 void Tarjan(int u,int id){
214     dfn[u]=low[u]=++dfs;
215     st.push(u);
216     for(int i=head[u];~i;i=edge[i].nx){
217         int v=edge[i].v;
218         if(id==(i^1)) continue;
219         if(!dfn[v]){
220             Tarjan(v,i);
221             low[u]=min(low[u],low[v]);
222             if(low[v]==dfn[v]){
223                 int cnt=0,x;
224                 do{
225                     x=st.top();st.pop();cnt+=a[x];
226                 }while(x!=v);
227                 ret=min(ret,abs(sum-2*cnt));
228                 a[u]+=cnt;
229             }
230         }else low[u]=min(low[u],dfn[v]);
231     }
232 }
233
234 //blocks=0,ebc表示第i个点属于哪一个双联通分量
235 //instack去除不知道会不会有事
236 stack<int>st;
237 int blocks,ebc[N];
238 int instack[N];
239
```

```
void Tarjan(int u,int id){
    low[u]=dfn[u]=++dfs;
    st.push(u);
    instack[u]=1;
    for(int i=head[u];~i;i=edge[i].nx){
        int v=edge[i].v;
        if(id==(i^1)) continue;
        if(!dfn[v]) {
            Tarjan(v, i);
            low[u] = min(low[u],low[v]);
        }else if(instack[v]&&dfn[v]<dfn[u]) low[u]=min(low[u],low[v]);
    }
    if(dfn[u]==low[u]){
        blocks++;
        int curr;
        do{
            curr=st.top();
            st.pop();
            instack[curr]=0;
            ebc[curr]=blocks;
        }while(curr!=u);
    }
}

点双连通分量 BCC
    对于一个连通图,如果任意两点至少存在两条“点不重复”的路径,则说图是点双连通的（即任意两条边都在一个简单环中）,点双连通的
极大子图称为点双连通分量。 通常来说,如果要求任意两条边在同一个简单环中,那么就是求点-双连通
    易知每条边属于一个连通分量，且连通分量之间最多有一个公共点，且一定是割点。
    无向连通图中割点一定属于至少两个BCC，非割点只属于一个BCC。
    注意：两个直接连接的点也是bcc,但是单个点不算bcc

//一个公园中有n个景点，景点之间通过无向的道路来连接，如果至少两个环公用一条路，路上的游客就会发生冲突；
//如果一条路不属于任何的环，这条路就没必要修，问，有多少路不必修，有多少路会发生冲突

//bcc存的是点双联通分量中的点，初始化bcc_cnt=-1，bccno为-1
vector<int>bcc[N];
int bcc_cnt,bccno[N];
stack<node>st;

void Tarjan(int u,int id){
    dfn[u]=low[u]=++dfs;
    for(int i=head[u];~i;i=edge[i].nx){
        int v=edge[i].v;
        if(id==(i^1)) continue;
        if(!dfn[v]){
            st.push(edge[i]);
            Tarjan(v,i);
            low[u]=min(low[u],low[v]);
            //判断桥
            if(low[v]==dfn[v]) ret++;
            int cnt=0;node x;
            //获得点双联通分量
            if(low[v]>=dfn[u]){
                bcc_cnt++; bcc[bcc_cnt].clear();
                do{
                    cnt++;
                    x=st.top(); st.pop();
                    if(bccno[x.u]!=bcc_cnt){
```

```
                        bcc[bcc_cnt].push_back(x.u);
                        bccno[x.u]=bcc_cnt;
                    }
                    if(bccno[x.v]!=bcc_cnt){
                        bcc[bcc_cnt].push_back(x.v);
                        bccno[x.v]=bcc_cnt;
                    }
                }while(x.u!=u||x.v!=v);
                if(bcc[bcc_cnt].size()<cnt) ret2+=cnt;
            }
        }else if(dfn[v]<dfn[u]){ //加入还没有加入过的边
            st.push(edge[i]);
            low[u]=min(low[u],dfn[v]);
        }
    }
}
```

4.7 二分匹配

```
1、匈牙利算法(O(n*m))

二分图最小顶点覆盖 = 二分图最大匹配;
最小覆盖要求用最少的点（X集合或Y集合的都行）让每条边都至少和其中一个点关联。

DAG图的最小路径覆盖 = 节点数（n）- 最大匹配数;
最小路径覆盖：用尽量少的不相交简单路径覆盖有向无环图G的所有结点。1~n匹配1~n的最大匹配数。

二分图最大独立集 = 节点数（n）- 最大匹配数;
二分图最大独立集要求从二分图中选出一些点，使这些点两两互不相邻。即没有独立集中任意两点没有边相连

最大匹配数：最大匹配的匹配边的数目

最小点覆盖数：选取最少的点，使任意一条边至少有一个端点被选择

最大独立数：选取最多的点，使任意所选两点均不相连

最小路径覆盖数：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长可以
    为 0（即单个点）。

int k,n,m,girl[N];
vector<int>g[N];
bool vis[N];
//n表示男生数，m表示女生数，girl[i]表示i女生搭档的男生
bool Match(int x){
    for(int i=0;i<g[x].size();i++){
        int y=g[x][i];
        if(!vis[y]){
            vis[y]=1;
            if(!girl[y]||Match(girl[y])){
                girl[y]=x; return true;
            }
        }
    }
    return false;
}

void gao(int n,int m){
    int ret=0;
```

```
    memset(girl,0, sizeof(int)*(m+1));
    for(int i=1;i<=n;i++){
        memset(vis,0, sizeof(bool)*(m+1));
        if(Match(i)) ret++;
    }
    printf("%d\n",ret);
}
```

4.8 第 k 最短路

```
1、给你一个无向图(有向也可)，可以将图中k个路的值变为0，求s->t的最短距离

输入:
//n,m,s,t,k
3 2 1 3 1
1 2 1
2 3 2

int n,m,s,t,k;
bool vis[N];
ll dp[1005][N];
struct edge{
    int v,nx;
    ll w;
}e[M];
int tot,head[N];
void add_edge(int u,int v,ll w){
    e[tot].v=v;e[tot].w=w;e[tot].nx=head[u];
    head[u]=tot++;
}

struct node{
    int u; ll w;
    bool operator<(const node&t)const{
        return w>t.w;
    }
};

void Dijkstra(int st,int ed,int k){
    memset(dp[k],0x3f, sizeof(ll)*(n+1));
    memset(vis,0, sizeof(bool)*(n+1));
    dp[k][st]=0;
    priority_queue<node>q;
    q.push(node{st,0});
    while(!q.empty()){
        node x=q.top();q.pop();
        if(vis[x.u]) continue;
        vis[x.u]=1;
        for(int i=head[x.u];~i;i=e[i].nx){
            int v=e[i].v; ll w=e[i].w;
            if(vis[v]) continue;
            if(k==0){
                if(dp[k][v]>dp[k][x.u]+e[i].w){
                    dp[k][v]=dp[k][x.u]+e[i].w;
                    q.push(node{v,dp[k][v]});
                }
            }else{
                ll len=min(dp[k-1][x.u],dp[k][x.u]+w);
```

```
                if(dp[k][v]>len){
                    dp[k][v]=len;
                    q.push(node{v,len});
                }
            }
        }
    }
}

int main(){
    int u,v,w;
    scanf("%d%d%d%d%d",&n,&m,&s,&t,&k);
    tot=0; memset(head,-1, sizeof(int)*(n+1));
    for(int i=1;i<=m;i++){
        scanf("%d%d%d",&u,&v,&w);
        add_edge(u,v,w);add_edge(v,u,w);
    }
    for(int i=0;i<=k;i++) Dijkstra(s,t,i);
    ll ret=INF;
    for(int i=0;i<=k;i++) ret=min(ret,dp[i][t]);
    printf("%lld\n",ret);
}
```

4.9 2-SAT

概念:
2-SAT，简单的说就是给出n个集合，每个集合有两个元素，已知若干个<a,b>，表示a与b矛盾（其中a与b属于不同的集合）。
然后从每个集合选择一个元素，判断能否一共选n个两两不矛盾的元素。显然可能有多种选择方案，一般题中只需要求出一种即可。

1、tarjan
假设有a1,a2和b1,b2两对，已知a1和b2间有矛盾，于是为了方案自洽，由于两者中必须选一个，所以我们就要拉两条有向边(a1,b1)和(b2,a2)表示选了a1则必须选b1，
选了b2则必须选a2才能够自洽。然后通过这样子建边我们跑一遍Tarjan SCC判断是否有一个集合中的两个元素在同一个SCC中，若有则输出不可能，否则输出方案。
构造方案只需要把几个不矛盾的 SCC 拼起来就好了。

寻求一组可行解:
当x所在的强连通分量的拓扑序在x'所在的强连通分量的拓扑序之后取x为真就可以了。在使用Tarjan算法缩点找强连通分量的过程中，已经为每组强连通分量标记好
顺序了——不过是反着的拓扑序。

2、爆搜模板,可以求字典序可行最优解

```
struct Twosat {
    int n;
    vector<int> g[N * 2];
    bool mark[N * 2];
    int s[N * 2], c;
    bool dfs(int x) {
        if (mark[x ^ 1]) return false;
        if (mark[x]) return true;
        mark[x] = true;
        s[c++] = x;
        for (int i = 0; i < (int)g[x].size(); i++)
            if (!dfs(g[x][i])) return false;
        return true;
    }
```

```
    void init(int n) {
        this->n = n;
        for (int i = 0; i < n * 2; i++) g[i].clear();
        memset(mark, 0, sizeof(mark));
    }
    void add_clause(int x, int y) {  // 这个函数随题意变化
        g[x].push_back(y ^ 1);         // 选了 x 就必须选 y^1
        g[y].push_back(x ^ 1);
    }
    bool solve() {
        for (int i = 0; i < n * 2; i += 2)
            if (!mark[i] && !mark[i + 1]) {
                c = 0;
                if (!dfs(i)) {
                    while (c > 0) mark[s[--c]] = false;
                    if (!dfs(i + 1)) return false;
                }
            }
        return true;
    }
}sat;

int main() {
    int x,y;
    while(scanf("%d%d",&n,&m)==2){
        sat.init(n);
        for(int i=1;i<=m;i++){
            scanf("%d%d",&x,&y);
            x--;y--;
            sat.add_clause(x,y);
        }
        if(sat.solve()){
            for(int i=0;i<2*n;i++){
                if(sat.mark[i]) printf("%d\n",i+1);
            }
        }else printf("NIE\n");
    }
}
```

建模情况:

1、(A,B)不能同时选: 选了A就要选B', 选了B就要选A', 所以要建立A->B',B->A'边

2、(A,B)不能同时不取: 选择了A'就只能选择B, 选择了B'就只能选择A,所以要建立A'->B,B'->A的边
   如要a|b==1,其中a和b为真假状态, 那么有三种建边情况
   1) a=b=1|0,那么会建边0->1或1->0
   2) a=1,b=0或a=0,b=1,那么会建边0->0,1->1
   3)上述方法

3、如果存在一个图, 图中点需要赋予0或1的值, 满足边a op b == c,其中op有and, or, xor
   有六种情况需要讨论:
   1) a and b == 1, 则需建边a=0->a=1,b=0->b=1。这里需要体会一下, 只需要两条即可
   2) a and b == 0, 则需建边a=1->b=0,b=1->a=0。
   3) a or b == 1, 则需建边a=0->b=1,b=0->a=1。
   4) a or b == 0, 则需建边a=1->a=0,b=1->b=0。同1情况
   5) a xor b == 1, 则需建边a=0->b=1,a=1->b=0,b=0->a=1,b=1->a=0。
   6) a xor b == 0, 则需建边a=0->b=0,b=0->a=0,a=1->b=1,b=1->a=1。

4、对于(a,b,c)，若a留下，则b，c回家，若b，c留下，a回家，对于(a,b)若a留下，b回家，若b留下，a回家，对于
有两种情况需要讨论：
1) a=1->b=0,c=0 b=1,c=1->a=0
2) a=0->b=1,b=0->a=1

5、有n个集合(a,b)，表示炸弹可放的两个位置，要求将n个炸弹都放置，求最大的半径长度
二分答案，通过二分的数据进行约束，创建2-sat

6、给出n个牛棚、两个特殊点S1,S2的坐标。S1、S2直连。牛棚只能连S1或S2，还有，某些牛棚只能连在同一个S，某些牛棚不能连在同一个S。
求使最长的牛棚间距离最小，距离是曼哈顿距离，使最大值最小。
二分答案，用2-sat判断是否可行
1.hate关系的a,b。 a->b^1,b->a^1,a^1->b,b^1->a
2.friend关系的a,b。 a->b,b->a,a^1->b^1,b^1->a^1
接下来的也要检查，因为引入参数，就是多了约束条件了
这四种情况就是i,j到达对方的所有情况了
3.dis[a]+dis[b]>limit a->b^1,b->a^1
4.dis[a^1]+dis[b^1]>limit a^1->b,b^1->a
5.dis[a]+dis[b^1]+tdis>limit a->b,b^1->a^1
6.dis[a^1]+dis[b]+tdis>limit a^1->b^1,b->a

7、两者（A，B）要么都取，要么都不取
建边：a->b,b->a,a^1->b^1,b^1->a^1

8、两者（A，A'）必取A
建边：a^1->a

4.10 LCA

LCA:
在一棵没有环的树上，每个节点肯定有其父亲节点和祖先节点，而最近公共祖先，就是两个节点在这棵树上深度最大的
公共的祖先节点。所以LCA主要是用来处理当两个点仅有唯一一条确定的最短路径时的路径。

1、Tarjan离线算法
什么是Tarjan(离线)算法呢？顾名思义，就是在一次遍历中把所有询问一次性解决，所以其时间复杂度是 (+)。
Tarjan算法的优点在于相对稳定，时间复杂度也比较居中，也很容易理解下面详细介绍一下Tarjan算法的基本思路：
1）选一个点为根节点，从根节点开始。
2）遍历该点u所有子节点v，并标记这些子节点v已被访问过。
3）若是v还有子节点，返回2，否则下一步。
4）合并v到u上。
5）寻找与当前点u有询问关系的点v。
6）若是v已经被访问过了，则可以确认u和v的最近公共祖先为v被合并到的父亲节点a。

代码：
```
//第一行包含三个正整数 N,M,S，分别表示树的结点个数、询问的个数和树根结点的序号。
//输出m次询问的公共祖先
//使用并查集查询结果，其中子节点必须并到父节点的祖先上
//注意并查集parent需要初始化
int n,m,s,ans[N],parent[N];
int ehead[N],etot;
struct edge{
    int v,nx;
}e[N<<1];
```

```
26 void add_edge(int u,int v){
27     e[etot].v=v; e[etot].nx=ehead[u];
28     ehead[u]=etot++;
29 }
30
31 int qhead[N],qtot;
32 struct query{
33     int v,id,nx;
34 }q[N<<1];
35
36 void add_query(int u,int v,int id){
37     q[qtot].v=v; q[qtot].nx=qhead[u]; q[qtot].id=id;
38     qhead[u]=qtot++;
39 }
40
41 bool vis[N];
42
43 int find(int x){
44     if(x!=parent[x]) parent[x]=find(parent[x]);
45     return parent[x];
46 }
47
48 void Merge(int x,int y){
49     int fx=find(x),fy=find(y);
50     if(fx!=fy){
51         parent[fy]=fx;
52     }
53 }
54
55 void Tarjan(int u){
56     vis[u]=1;
57     for(int i=ehead[u];i;i=e[i].nx){
58         int v=e[i].v;
59         if(vis[v]) continue;
60         Tarjan(v);
61         Merge(u,v);
62     }
63     for(int i=qhead[u];i;i=q[i].nx){
64         int v=q[i].v;
65         if(vis[v]){
66             ans[q[i].id]=find(v);
67         }
68     }
69 }
70
71 int main() {
72     int u,v;
73     scanf("%d%d%d",&n,&m,&s);
74     etot=qtot=1;
75     for(int i=1;i<n;i++){
76         scanf("%d%d",&u,&v);
77         add_edge(u,v); add_edge(v,u);
78     }
79     for(int i=1;i<=m;i++){
80         scanf("%d%d",&u,&v);
81         add_query(u,v,i); add_query(v,u,i);
82     }
83     memset(vis,0, sizeof(bool)*(n+1));
84     for(int i=1;i<=n;i++) parent[i]=i;
```

```
    Tarjan(s);
    for(int i=1;i<=m;i++) printf("%d\n",ans[i]);
}

2、树上倍增LCA

const int MAX_DEP = 20;

// 倍增2^k的父亲
int fa[N][MAX_DEP];
int dep[N];

int n,m,s;
int head[N],tot=1;
struct edge{
    int v,nx;
}e[N<<1];

void add_edge(int u,int v){
    e[tot].v=v; e[tot].nx=head[u];
    head[u]=tot++;
}

void lineFa(int u,int v){
    fa[u][0]=v;
    for(int i=1;i<MAX_DEP;i++)
        v=fa[u][i]=fa[v][i-1];
}

void dfs(int u,int pre){
    for(int i=head[u];i;i=e[i].nx){
        int v=e[i].v;
        if(v==pre) continue;
        dep[v]=dep[u]+1;
        lineFa(v,u);
        dfs(v,u);
    }
}

int LCA(int u,int v){
    if(dep[u]>dep[v]) swap(u,v);
    int hu=dep[u],hv=dep[v];
    int tu=u,tv=v;
    for(int det=hv-hu,i=0;det;det>>=1,i++){
        if(det&1) tv=fa[tv][i];
    }
    if(tu==tv) return tu;
    for(int i=MAX_DEP-1;i>=0;i--){
        if (fa[tu][i] == fa[tv][i]) {
            continue;
        }
        tu = fa[tu][i];
        tv = fa[tv][i];
    }
    return fa[tu][0];
}

int main() {
    int u,v;
```

```
    scanf("%d%d%d",&n,&m,&s);
    tot=1;
    for(int i=1;i<n;i++){
        scanf("%d%d",&u,&v);
        add_edge(u,v); add_edge(v,u);
    }
    dep[s]=0;
    dfs(s,0);
    for(int i=1;i<=m;i++){
        scanf("%d%d",&u,&v);
        printf("%d\n",LCA(u,v));
    }
}

3、st表查询
//注意调用LCA_init初始化
int n,m,rt,head[N],tot;
int dfn[N],pos[N],rmq[N],dno;
struct edge{
    int v,nx;
}e[N<<1];
void add_edge(int u,int v){
    e[tot].v=v; e[tot].nx=head[u];
    head[u]=tot++;
}

struct ST {
    int k2[21], st[21][N], Log[N];
    void init_st(int n) {
        k2[0] = 1;
        for (int i = 1; i <= 20; i++) k2[i] = 2 * k2[i - 1];
        Log[0] = -1; for (int i = 1; i <=n ; i++) Log[i] = Log[i / 2] + 1;
        for (int i = 1; i <= n; i++) st[0][i] = i;
        for (int i = 1; i <= Log[n]; i++) {
            for (int j = 1; j + k2[i] - 1 <= n; j++) {
                st[i][j] = (rmq[st[i - 1][j]]<rmq[st[i - 1][j + k2[i - 1]]]) ?
                           st[i - 1][j]:st[i - 1][j + k2[i - 1]];
            }
        }
    }
    int query_min(int x, int y) {
        int len = log2(y - x + 1);
        return (rmq[st[len][x]]<rmq[st[len][y - k2[len] + 1]]) ?
               st[len][x]:st[len][y - k2[len] + 1];
    }
}st;

void dfs(int u,int pre,int dep){
    dfn[++dno]=u;
    rmq[dno]=dep;
    pos[u]=dno;
    for(int i=head[u];i;i=e[i].nx){
        int v=e[i].v;
        if(v==pre) continue;
        dfs(v,u,dep+1);
        dfn[++dno]=u;
        rmq[dno]=dep;
    }
}
```

```
203
204 void LCA_init(int root,int n){
205     dno=0;
206     dfs(root,root,0);
207     st.init_st(2*n-1);
208 }
209
210 int LCA(int u,int v){
211     int pu=pos[u],pv=pos[v];
212     if(pu>pv) swap(pu,pv);
213     return dfn[st.query_min(pu,pv)];
214 }
215
216 int main() {
217     int u,v;
218     scanf("%d%d%d",&n,&m,&rt);
219     tot=1; memset(head,0, sizeof(int)*(n+1));
220     for(int i=1;i<n;i++){
221         scanf("%d%d",&u,&v);
222         add_edge(u,v); add_edge(v,u);
223     }
224     LCA_init(rt,n);
225     for(int i=1;i<=m;i++) {
226         scanf("%d%d", &u, &v);
227         printf("%d\n", LCA(u, v));
228     }
229 }
230
231 4、朴素查询
232 while(q--){
233     scanf("%d%d",&u,&v);
234     u=find(ebc[u]); v=find(ebc[v]);
235     if(u==v){
236         printf("%d\n",ret);
237         continue;
238     }
239     if(dep[u]<dep[v]) swap(u,v);
240     int i=u,j=v;
241     while(parent[i]!=parent[j]){
242         if(dep[parent[i]]<dep[parent[j]]) swap(i,j);
243         ret--; i=find(f[i]);
244     }
245     printf("%d\n",ret);
246     while(parent[u]!=parent[v]){
247         if(dep[parent[u]]<dep[parent[v]]) swap(u,v);
248         parent[u]=i; u=find(f[u]);
249     }
250 }
```

4.11 欧拉路

1 连通图:
2 在图论中，连通图基于连通的概念。在一个无向图 G 中，若从顶点i到顶点j有路径相连（当然从j到i也一定有路径）
3 ，则称i和j是连通的。如果 G 是有向图，那么连接i和j的路径中所有的边都必须同向。如果图中任意两点都是连通的，那
4 么图被称作连通图。如果此图是有向图，则称为强连通图（注意：需要双向都有路径）。图的连通性是图的基本性质。
5

定义：
欧拉路：欧拉路是指从图中任意一个点开始到图中任意一个点结束的路径，并且通过图中每条边，且只通过一次。
欧拉回路：欧拉回路是指起点和终点相同的欧拉路。

无向图是否具有欧拉路或回路的判定：
    欧拉路：图连通，所有点度都是偶数，或者恰好有两个点度是奇数，则有欧拉路。若有奇数点度，则奇数点度点一定是欧
拉路的起点和终点，否则可取任意一点作为起点。
    欧拉回路：图连通，图中所有节点度均为偶数

有向图是否具有欧拉路或回路的判定：
    欧拉路：图连通，除2个端点外其余节点人度=出度；1个端点人度比出度大1；一个端点人度比出度小1，取出度大者为起点，人度大者为终点。
或 所有节点人度等于出度
    欧拉回路：图连通，所有节点人度等于出度

对于Hierholzers算法，前提是假设图G存在欧拉回路，即有向图任意点的出度和人度相同。从任意一个起始点v开始遍历，直到再次到达点v，
即寻找一个环，这会保证一定可以到达点v，因为遍历到任意一个点u，由于其出度和人度相同，故u一定存在一条出边，所以一定可以到达v。
将此环定义为C，如果环C中存在某个点x，其有出边不在环中，则继续以此点x开始遍历寻找环C'，将环C、C'连接起来也是一个大环，如此往
复，直到图G中所有的边均已经添加到环中。

```
//有向图欧拉回路打印路径模板
//因为有向图欧拉回路的性质，其实st中正序或倒叙输出都没问题，都要回到最初点u=1
void dfs(int u){
    for(int i=head[u];~i;i=e[i].nx){
        int v=e[i].v;
        if(vis[i]) continue;
        vis[i]=1;
        head[u]=i;
        dfs(v);
        i=head[u];
    }
    st[++st[0]]=u;
}

//非递归版
int st[N];
int syst[N*10],systop;

void dfs(int u){
    systop=0;
    syst[++systop]=1; //初始点进入
    while(systop>0){
        int x=syst[systop],i=head[x];
        while((~i)&&vis[i]) i=e[i].nx;
        if(~i){
            syst[++systop]=e[i].v;
            vis[i]=1;
            head[x]=e[i].nx;
        }else{
            systop--,st[++st[0]]=x;
        }
    }
}

//无向图欧拉回路打印路径模板，边或点
```

```cpp
void dfs(int u){
    for(int i=head[u];~i;i=e[i].nx){
        int v=e[i].v;
        if(vis[i]) continue;
        vis[i]=vis[i^1]=1; //有向图只用禁掉一条，无向禁两条，具体禁的边看构图而定
        dfs(v);
        //st里边存边，外部存点，st[st[0]]->st[1]输出就是路径结果
    }
    st[++st[0]]=u;
}
```

问题一:
一个无向图图中（不是所有点联通，无重边，无自环），求问最少要几笔才能将所有边画到，孤立点不用画。
考虑无向图欧拉路三种情况:
1、若只存在一个点，则答案贡献为0，
2、若奇数入度点为0，则ans+=1
3、若奇数入度点为cnt,则ans+=cnt/2

问题二:
给n个字符串，要求排个序，使得si字符串的尾部是si+1字符串的首部，求问是否存在这种排序序列
思路:
初始想法就是在一个带有字符串关系之间的图中找到一个合法结果，显然不现实，因为n<=1e5。那么可以设立
in[26],out[26],vis[26],parent[26]数组，统计各字符串的出入读，并用并查集进行关系合并。若发现只有一个
i==parent[i],说明只有一个头，那么可能存在合法序列。否则不行。然后根据有向图欧拉路的欧拉路定义，只有满足
该条件才能存在合法序列。
注意:
1.判断是否所有字符串连通 2.注意有向还是无向，从而通过定义解决问题 3、对于给定字符串，如果顺序不能改变，
那么首字母为out++，尾字母为in++

5 数学

5.1 BM

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=mod; for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}
    return res;}
// head

ll n;
namespace linear_seq {
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];

    vector<int> Md;
    void mul(ll *a,ll *b,int k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
        ll ans=0,pnt=0;
        int k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
```

```
        rep(n,0,SZ(s)) {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n) {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n) {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};

int main() {
    /*push_back 进去前 8~10 项左右、最后调用 gao 得第 n 项*/
    vector<int>v;
    v.push_back(3);
    v.push_back(9);
    v.push_back(20);
    v.push_back(46);
    v.push_back(106);
    v.push_back(244);
    v.push_back(560);
    v.push_back(1286);
    v.push_back(2956);
    v.push_back(6794);
    int nCase;
    scanf("%d", &nCase);
    while(nCase--){
        scanf("%lld", &n);
        printf("%lld\n",1LL * linear_seq::gao(v,n-1) % mod);
    }
}
```

5.2 gcd、ex-gcd

```
1、gcd
ll gcd(ll a,ll b){
    return b>0?gcd(b,a%b):a;
}

 斐波那契数列的最大公约数定理：gcd(F(m),F(n))=F(gcd(m,n))
 F[0]=0,F[1]=1...

```

```
2、ex-gcd
void extend_gcd(ll a, ll b, ll &x, ll &y) {
    if (!b){
        x = 1, y = 0;
        return;
    }
    else{
        extend_gcd(b, a % b, y, x);
        y -= x * (a / b);
        return;
    }
}

ll inv(ll a, ll n) {
    ll x, y;
    extend_gcd(a,n,x,y);
    x = (x % n + n) % n;
    return x;
}
```

5.3 拉格朗日插值法

```
/*
测试用例 （函数）x^3-2*x+7
a[]={7,6,11,28,63}
第0-4项和 115
最高次项3，代入项数4
*/

ll powmod(ll a, ll b) { ll res = 1; a %= mod; for (; b; b >>= 1) { if (b & 1)res = res
    * a % mod; a = a * a % mod; }return res; }

namespace polysum {
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
    const int D = 1e5;
    ll a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
    //函数用途：给出数列的（d+1）项，其中d为最高次方项
    //求出数列的第n项，数组下标从0开始
    ll calcn(int d, ll* a, ll n) { // a[0].. a[d]  a[n]
        if (n <= d) return a[n];
        p1[0] = p2[0] = 1;
        rep(i, 0, d + 1) {
            ll t = (n - i + mod) % mod;
            p1[i + 1] = p1[i] * t % mod;
        }
        rep(i, 0, d + 1) {
            ll t = (n - d + i + mod) % mod;
            p2[i + 1] = p2[i] * t % mod;
        }
        ll ans = 0;
        rep(i, 0, d + 1) {
            ll t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
            if ((d - i) & 1) ans = (ans - t + mod) % mod;
            else ans = (ans + t) % mod;
        }
        return ans;
    }
```

```
36      void init(int M) {
37          f[0] = f[1] = g[0] = g[1] = 1;
38          rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
39          g[M + 4] = powmod(f[M + 4], mod - 2);
40          per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod;
41      }
42      //函数用途：给出数列的（m+1）项，其中m为最高次方
43      //求出数列的前（n-1）项的和
44      ll polysum(ll m, ll* a, ll n) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]
45          ll b[D];
46          for (int i = 0; i <= m; i++) b[i] = a[i];
47          b[m + 1] = calcn(m, b, m + 1);
48          rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
49          return calcn(m + 1, b, n - 1);
50      }
51      ll qpolysum(ll R, ll n, ll* a, ll m) { // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
52          if (R == 1) return polysum(n, a, m);
53          a[m + 1] = calcn(m, a, m + 1);
54          ll r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;
55          h[0][0] = 0; h[0][1] = 1;
56          rep(i, 1, m + 2) {
57              h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
58              h[i][1] = h[i - 1][1] * r % mod;
59          }
60          rep(i, 0, m + 2) {
61              ll t = g[i] * g[m + 1 - i] % mod;
62              if (i & 1) p3 = ((p3 - h[i][0] * t) % mod + mod) % mod, p4 = ((p4 - h[i][1]
     * t) % mod + mod) % mod;
63              else p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
64          }
65          c = powmod(p4, mod - 2) * (mod - p3) % mod;
66          rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
67          rep(i, 0, m + 2) C[i] = h[i][0];
68          ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
69          if (ans < 0) ans += mod;
70          return ans;
71      }
72  } // polysum::init();
73
74
75  ll b[N];
76  int n,m;
77  int l,r;
78
79  int main(){
80      int t;
81      scanf("%d",&t);
82      while(t--){
83          scanf("%d%d",&n,&m);
84          polysum::init(n+10);
85          for(int i=0;i<=n;i++)
86              scanf("%lld",&b[i]);
87          while(m--){
88              scanf("%d%d",&l,&r);
89              printf("%lld\n",((polysum::polysum(n,b,r+1)-polysum::polysum(n,b,l))%mod+
    mod)%mod);
90          }
91      }
92  }
```

5.4 素数

```
1、素数筛
//Mark中标记为true的为合数
int prime[N];
bool Mark[N];
int cnt=0;

void Prime(int n){
    for(int i=2;i<=n;i++){
        if(Mark[i]==0)
            prime[cnt++]=i;
        for(int j=0;j<cnt&&prime[j]*i<=n;j++){
            Mark[i*prime[j]]=1;
            if(i%prime[j]==0)
                break;
        }
    }
}

//获得[2,n]的所有最小质因子和，1不为质因子
ll res=0;
int n,cnt;
int prime[N];
bool Mark[N];

void Prime(int n){
    for(int i=2;i<=n;i++){
        if(!Mark[i])
            res+=prime[cnt++]=i;
        for(int j=0,e=n/i;j<cnt&&prime[j]<=e;j++){
            Mark[i*prime[j]]=1;
            res+=prime[j];
            if(i%prime[j]==0) break;
        }
    }
}

//求[2,n]内所有数能分解出的合数因子,复杂度nlogn
void init(int n){
    Prime(n);
    tot=0; memset(head,-1, sizeof(head));
    for(int v=2;v<=n;v++){
        if(!Mark[v]) continue; //若v为合数则可以加入答案
        for(int u=v;u<=n;u+=v){
            add_edge(u,v);
        }
    }
}

//若一个数满足n=ab,a和b不能被平方数整除，除了1，设f(n)为满足对数，比如f(4)=1,2*2,f(6)
    =4,1*6,6*1,2*3,3*2
//求[1,n]之间f(i)的和，其中n<=2e7
void Prime(int n){
    dp[1]=1;//必须放在第一个
    for(int i=2;i<=n;i++){
        if(Mark[i]==0) {
            prime[cnt++]=i;dp[i]=2;
        }
```

```
        for(int j=0;j<cnt&&prime[j]*i<=n;j++){
            int di=i*prime[j];
            Mark[di]=1;
            if(i%prime[j]) dp[di]=dp[i]*2;
            else{
                if((i/prime[j])%prime[j]==0) dp[di]=0;
                else dp[di]=dp[i/prime[j]];
                break;
            }
        }
    }
    for(int i=2;i<=n;i++) dp[i]+=dp[i-1];
}

2、单点判断
bool isPrime(ll num){
    if (num == 2 || num == 3)
        return true;
    if (num % 6 != 1 && num % 6 != 5)
        return false;
    for (ll i = 5; i*i <= num; i += 6)
        if (num % i == 0 || num % (i+2) == 0)
            return false;
    return true;
}


3、一个素数为P,它之前的素数为Q,求Q!%P的值
int cnt=0;
ll p,q;

bool isPrime(ll num){
    if (num == 2 || num == 3)
        return true;
    if (num % 6 != 1 && num % 6 != 5)
        return false;
    for (ll i = 5; i*i <= num; i += 6)
        if (num % i == 0 || num % (i+2) == 0)
            return false;
    return true;
}

void extend_gcd(ll a, ll b, ll &x, ll &y) {
    if (!b){
        x = 1, y = 0;
        return;
    }
    else{
        extend_gcd(b, a % b, y, x);
        y -= x * (a / b);
        return;
    }
}

ll inv(ll a, ll n) {
    ll x, y;
    extend_gcd(a,n,x,y);
    x = (x % n + n) % n;
    return x;
```

```
}

int main() {
    int t;
    scanf("%d",&t);
    while(t--){
        scanf("%lld",&p);
        q=p-1;
        while(!isPrime(q))
            q--;
        ll top = p-q-1;
        if(top==0)
            top=1;
        ll res=1;
        for(ll i=1;i<=top;i++){
            res=res*i%p;
        }
        res = inv(res,p);
        printf("%lld\n",res);
    }
}

4、
LL Mult_Mod(LL a,LL b,LL m)//res=(a*b)%m
{
    a%=m;
    b%=m;
    LL res=0;
    while(b)
    {
        if(b&1)
            res=(res+a)%m;
        a=(a<<=1)%m;
        b>>=1;
    }
    return res%m;
}
LL Pow_Mod(LL a, LL b, LL m)//res=(a^b)%m
{
    LL res=1;
    LL k=a;
    while(b)
    {
        if((b&1))
            res=Mult_Mod(res,k,m)%m;

        k=Mult_Mod(k,k,m)%m;
        b>>=1;
    }
    return res%m;
}

bool Witness(LL a,LL n,LL x,LL sum)
{
    LL judge=Pow_Mod(a,x,n);
    if(judge==n-1||judge==1)
        return 1;

    while(sum--)
```

```
175     {
176         judge=Mult_Mod(judge,judge,n);
177         if(judge==n-1)
178             return 1;
179     }
180     return 0;
181 }
182
183 bool Miller_Rabin(LL n)
184 {
185     if(n<2)
186         return 0;
187     if(n==2)
188         return 1;
189     if((n&1)==0)
190         return 0;
191
192     LL x=n-1;
193     LL sum=0;
194     while(x%2==0)
195     {
196         x>>=1;
197         sum++;
198     }
199
200
201     int times=20;
202     for(LL i=1;i<=times;i++)
203     {
204         LL a=rand()%(n-1)+1;//取与p互质的整数a
205         if(!Witness(a,n,x,sum))//费马小定理的随机数检验
206             return 0;
207     }
208     return 1;
209 }
210 LL GCD(LL a,LL b)
211 {
212     return b==0?a:GCD(b,a%b);
213 }
214 LL Pollard_Rho(LL n,LL c)//寻找一个因子
215 {
216     LL i=1,k=2;
217     LL x=rand()%n;//产生随机数x0(并控制其范围在1 ~ x-1之间)
218     LL y=x;
219     while(1)
220     {
221         i++;
222         x=(Mult_Mod(x,x,n)+c)%n;
223         LL gcd=GCD(y-x,n);
224
225         if(gcd<0)
226             gcd=-gcd;
227
228         if(gcd>1&&gcd<n)
229             return gcd;
230
231         if(y==x)
232             return n;
233
```

```
        if(i==k)
        {
            y=x;
            k<<=1;
        }
    }
}

int total;//因子的个数
LL factor[N];//存储所有因子的数组，无序的
void Find_fac(LL n)//对n进行素因子分解，存入factor
{
    if(Miller_Rabin(n))//是素数就把这个素因子存起来
    {
        factor[++total]=n;
        return;
    }

    long long p=n;
    while(p>=n)//值变化，防止陷入死循环k
        p=Pollard_Rho(p,rand()%(n-1)+1);

    Find_fac(n/p);
    Find_fac(p);
}
```

5.5 高斯消元

```
1.解同余方程模板
ll a[N][N],x[N];
inline ll gcd(ll a,ll b) {
    return b ? gcd(b, a % b) : a;
}

inline ll lcm(ll a,ll b) {
    return a/gcd(a,b)*b;
}

ll inv(ll a,ll p){
    if(a == 1) return 1;
    return inv(p%a,p)*(p-p/a)%p;
}

int Gauss(int equ,int var) {
    int max_r, col, k;
    for (k = 0, col = 0; k < equ && col < var; k++, col++) {
        max_r = k;
        for(int i = k+1; i < equ;i++)
            if(abs(a[i][col]) > abs(a[max_r][col]))
                max_r = i;
        if(a[max_r][col] == 0){
            k--;
            continue;
        }
        if(max_r != k)
            for(int j = col; j < var+1;j++)
                swap(a[k][j],a[max_r][j]);
        for(int i = k+1;i < equ;i++) {
```

```
            if (a[i][col] != 0) {
                ll LCM = lcm(abs(a[i][col]),abs(a[k][col]));
                ll ta = LCM/abs(a[i][col]);
                ll tb = LCM/abs(a[k][col]);
                if(a[i][col]*a[k][col] < 0)tb = -tb;
                for(int j = col;j < var+1;j++)
                    a[i][j] = ((a[i][j]*ta - a[k][j]*tb)%mod + mod)%mod;
            }
        }
    }
    for(int i = k;i < equ;i++)
        if(a[i][col] != 0)
            return -1;//无解
    if(k < var) return var-k;//多解
    for(int i = var-1;i >= 0;i--){
        ll temp = a[i][var];
        for(int j = i+1; j < var;j++) {
            if (a[i][j] != 0) {
                temp -= a[i][j] * x[j];
                temp = (temp % mod + mod) % mod;
            }
        }
        x[i] = (temp*inv(a[i][i],mod))%mod;
    }
    return 0;
}

2.
const double EPS = 1e-9;
inline int sign(double x){return (x>EPS)-(x<-EPS);}
double A[250][250];

bool gauss(int n){
    int i,j,k,r;
    for(i=0;i<n;i++){
        //选一行与r与第i行交换，提高数据值的稳定性
        r=i;
        for(j=i+1;j<n;j++)
            if(fabs(A[j][i]) > fabs(A[r][i]))r=j;
        if(r!=i)for(j=0;j<=n;j++)swap(A[r][j],A[i][j]);
        //i行与i+1~n行消元
        /*  for(k=i+1;k<n;k++){   //从小到大消元，中间变量f会有损失
              double f=A[k][i]/A[i][i];
              for(j=i;j<=n;j++)A[k][j]-=f*A[i][j];
          }*/
        for(j=n;j>=i;j--){   //从大到小消元，精度更高
            for(k=i+1;k<n;k++)
                A[k][j]-=A[k][i]/A[i][i]*A[i][j];
        }
    }
    //判断方程时候有解
    for(i=0;i<n;i++)if(sign(A[i][i])==0)return 0;
    //回代过程
    for(i=n-1;i>=0;i--){
        for(j=i+1;j<n;j++)
            A[i][n]-=A[j][n]*A[i][j];
        A[i][n]/=A[i][i];
    }
    return 1;
```

```
}
```

5.6 几何基础模板

```
class Point{
public:
    double x, y;
    Point(double x = 0,double y = 0) :x(x), y(y) {}
    Point operator + (Point a){
        return Point(x + a.x, y + a.y);
    }
    Point operator - (Point a){
        return Point(x - a.x, y - a.y);
    }
    bool operator < (const Point& a) const{
        if (x == a.x) return y < a.y;
        return x < a.x;
    }
    bool operator == (Point a){
        if (x == a.x && y == a.y) return true;
        return false;
    }
    double abs(void){
        return sqrt(x * x + y * y);
    }
};

typedef Point Vector;

//叉积
double cross(Vector a, Vector b){
    return a.x * b.y - a.y * b.x;
}

//点积
double dot(Vector a, Vector b){
    return a.x * b.x + a.y * b.y;
}

//判断方向
bool isclock(Point p0, Point p1, Point p2){
    Vector a = p1 - p0;
    Vector b = p2 - p0;
    if (cross(a, b) < 0) return true;
    return false;
}

typedef vector<Point> Polygon;

//求凸包
Polygon andrewScan(Polygon s) {
    Polygon u, l;
    if (s.size() < 3) return s;
    sort(s.begin(), s.end());
    u.push_back(s[0]);
    u.push_back(s[1]);
    l.push_back(s[s.size() - 1]);
    l.push_back(s[s.size() - 2]);
```

```
55      for (int i = 2; i < s.size(); i++){
56          for (int n = u.size(); n >= 2 && isclock(u[n - 2], u[n - 1], s[i]) != true; n
        --)
57              u.pop_back();
58          u.push_back(s[i]);
59      }
60      for (int i = s.size() - 3; i >= 0; i--){
61          for (int n = l.size(); n >= 2 && isclock(l[n - 2], l[n - 1], s[i]) != true; n
        --)
62              l.pop_back();
63          l.push_back(s[i]);
64      }
65      for (int i = 1; i < u.size() - 1; i++)
66          l.push_back(u[i]);
67      return l;
68  }
69
70  //判断符号
71  int signal(double x){
72      if(fabs(x)<eps)
73          return 0;
74      else
75          return x<0?-1:1;
76  }
77
78  //判断线段相交
79  bool segmentCross(Point a,Point b,Point c,Point d){
80  //快速排斥实验
81      if(max(c.x,d.x)<min(a.x,b.x)||max(a.x,b.x)<min(c.x,d.x)||max(c.y,d.y)<min(a.y,b.y)
        ||max(a.y,b.y)<min(c.y,d.y)){
82          return false;
83      }
84      //跨立实验
85      if(cross(a-d,c-d)*cross(b-d,c-d)>0||cross(d-b,a-b)*cross(c-b,a-b)>0){
86          return false;
87      }
88      return true;
89  }
90
91  //得到多边形面积
92  double getArea(Polygon s){
93      double sum=0;
94      double x1,y1,x2,y2;
95      int n = s.size()-1;
96      for(int i=1;i<=n-1;i++){
97          x1=s[i].x-s[0].x;
98          y1=s[i].y-s[0].y;
99          x2=s[i+1].x-s[0].x;
100         y2=s[i+1].y-s[0].y;
101         sum+=(x1*y2-x2*y1)/2;
102     }
103     return fabs(sum);
104 }
105
106 //判断点是否存在于多边形中
107 bool isOk(Point x,double area,Polygon s){
108     double sum=0;
109     int n = s.size();
110     for(int i=0;i<n;i++){
```

```
        Point &y = s[i],&z = s[(i+1)%n];
        sum+=fabs(cross(y-x,z-x))/2;
    }
    return fabs(sum-area)<=eps;
}


已知平面三个点，求外接圆圆心
double x=((y2-y1)*(y3*y3-y1*y1+x3*x3-x1*x1)-(y3-y1)*(y2*y2-y1*y1+x2*x2-x1*x1))/(2*(x3-x1)*(y2-y1)-2*((x2-x1)*(y3-y1)));
double y=((x2-x1)*(x3*x3-x1*x1+y3*y3-y1*y1)-(x3-x1)*(x2*x2-x1*x1+y2*y2-y1*y1))/(2*(y3-y1)*(x2-x1)-2*((y2-y1)*(x3-x1)));
printf("%.3lf %.3lf\n",x,y);


如何判断三角形是钝角、直角、还是锐角三角形
1、设c为最长的边
    若a^2+b^2<c^2,则为钝角三角形
    若a^2+b^2=c^2,则为直角三角形
    若a^2+b^2>c^2,则为锐角三角形
2、三点中，若存在点乘<0，则说明存在钝角
//i为顶点
bool check(int i,int j,int k){
    return ((p[i].x-p[j].x)*(p[i].x-p[k].x)+(p[i].y-p[j].y)*(p[i].y-p[k].y))<0;
}

点乘是向量的内积，叉乘是向量的外积
点乘，也叫数量积。结果是一个向量在另一个向量方向上投影的长度，是一个标量。
叉乘，也叫向量积。结果是一个和已有两个向量都垂直的向量。

在一条直线上，同向是叉乘为0，点乘为正，反向为叉乘为0，点乘为负

向量的点乘:a * b，x1*x2+y1*y2
公式：a * b = |a| * |b| * cos□  点乘又叫向量的内积、数量积，是一个向量和它在另一个向量上的投影的长度的乘积;
是标量。 点乘反映着两个向量的“相似度”，两个向量越“相似”，它们的点乘越大。

向量的叉乘：a ∧ b，x1*y2-x2*y1
a ∧ b = |a| * |b| * sin□  向量积被定义为： 模长： （在这里□ 表示两向量之间的夹角(共起点的前提下)（0° ≤ □ ≤ 180°），
它位于这两个矢量所定义的平面上。） 方向：a向量与b向量的向量积的方向与这两个向量所在平面垂直，且遵守右手定则。
（一个简单的确定满足“右手定则”的结果向量的方向的方法是这样的：若坐标系是满足右手定则的，当右手的四指从a以不超过180度的转角转向b时，
竖起的大拇指指向是c的方向。c = a ∧ b）

3、现有一个边长为正整数的三角形，问能否以其三个顶点为圆心画三个圆，使三个圆两两外切
    只要满足a+b>c，则必定有解，且结果为(a+b-c)/2,(a+c-b)/2,(b+c-a)/2;
```

5.7 大数模板

```
0、快速幂
ll quick(ll a,ll b){
    ll ret=1;a%=mod;
    while(b){
        if(b&1) ret=ret*a%mod;
        b>>=1;
        a=a*a%mod;
```

```
    }
    return ret;
}

1、求phi模板
ll phi(ll m) {
    ll ans = 1;
    for (ll i = 2; i*i <= m; i++) {
        if (m%i == 0) {
            m /= i;
            ans *= i - 1;
            while (m%i == 0) {
                m /= i;
                ans *= i;
            }
        }
    }
    if (m > 1) ans *= m - 1;
    return ans;
}

2、大数取模模板
ll Mod(string a,ll b){
    ll len=a.length()-1;
    ll ans=0;
    for(int i=0;i<=len;i++)
        ans=(ans*10+(a[i]-'0')%b)%b;
    return ans;
}

3、大数相乘模板
string Mul(string s,int x){
    reverse(s.begin(),s.end());
    int cmp=0;
    for(int i=0;i<s.size();i++){
        cmp=(s[i]-'0')*x+cmp;
        s[i]=(cmp%10+'0');
        cmp/=10;
    }
    while(cmp){
        s+=(cmp%10+'0');
        cmp/=10;
    }
    reverse(s.begin(),s.end());
    return s;
}

4、大数相加模板:
string sum(string s1,string s2){
    if(s1.length()<s2.length()) swap(s1,s2);
    int i,j;
    for(i=s1.length()-1,j=s2.length()-1;i>=0;i--,j--){
        s1[i]=char(s1[i]+(j>=0?s2[j]-'0':0));   //注意细节
        if(s1[i]-'0'>=10){
            s1[i]=char((s1[i]-'0')%10+'0');
            if(i) s1[i-1]++;
            else s1='1'+s1;
        }
    }
```

```
    return s1;
}

5、__int128 2^128次使用,只能在Linux下使用
inline __int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch<'0' || ch>'9') {
        if (ch == '-')
            f = -1;
        ch = getchar();
    }
    while (ch >= '0'&&ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}

inline void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9)
        print(x / 10);
    putchar(x % 10 + '0');
}

其他时候可以使用printf,cin,cout


6、快速乘
//O(1)快速乘
inline LL quick_mul(LL x,LL y,LL MOD){
    x=x%MOD,y=y%MOD;
    return ((x*y-(LL)(((long double)x*y+0.5)/MOD)*MOD)%MOD+MOD)%MOD;
}
//O(log)快速乘
inline LL quick_mul(LL a,LL n,LL m){
    LL ans=0;
    while(n){
        if(n&1) ans=(ans+a)%m;
        a=(a<<1)%m;
        n>>=1;
    }
    return ans;
}

7、高精度相加
struct BigInteger {
    static const int BASE = 10000; //高进制
    static const int WIDTH = 4; //高进制位数
    vector<int>s;
    BigInteger() {}
    BigInteger(long long num) { // 构造函数
        *this = num;
    }
    //赋值
```

```
126     BigInteger operator = (long long num) {
127         s.clear();
128         do {
129             s.push_back(num%BASE);
130             num /= BASE;
131         } while (num > 0);
132         return *this;
133     }
134     //+
135     BigInteger operator + (BigInteger& b) {
136         BigInteger c;
137         c.s.resize(max(s.size(), b.s.size()) + 1);
138         for (int i = 0; i < c.s.size() - 1; i++) {
139             int tmp1, tmp2;
140             if (i >= s.size())tmp1 = 0;
141             else tmp1 = s[i];
142             if (i >= b.s.size())tmp2 = 0;
143             else tmp2 = b.s[i];
144             c.s[i] = tmp1 + tmp2;
145         }
146         for (int i = 0; i < c.s.size() - 1; i++) {
147             c.s[i + 1] += c.s[i] / BASE;
148             c.s[i] %= BASE;
149         }
150         while (c.s.back() == 0 && c.s.size() > 1)c.s.pop_back();
151         return c;
152     }
153     void operator += (BigInteger& b) {
154         *this = *this + b;
155     }
156 };
157
158 BigInteger dp[55][265];
159
160 ostream& operator << (ostream& output, const BigInteger& x) {
161     output << x.s.back();
162     for (int i = x.s.size() - 2; i >= 0; i--) {
163         char buf[20];
164         sprintf(buf, "%04d", x.s[i]);
165         for (int j = 0; j < strlen(buf); j++)output << buf[j];
166     }
167     return output;
168 }
```

5.8 组合数学

```
1  1、打表求组合数
2  c[n][m],n>=m
3  for(int i=0;i<=n;i++){
4      c[i][0]=1;
5      for(int j=1;j<=i;j++){
6          c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
7      }
8  }
9
10 2、预处理，调用C(int n, int m)
11 ll dp[N], fac[N], inv[N];
12
```

```
ll quick(ll a, ll b) {
    ll res = 1;
    while (b) {
        if (b & 1) res = res * a % mod;
        b >>= 1;
        a = a * a % mod;
    }
    return res;
}

void init(int n) {
    fac[0] = 1, fac[1] = 1;
    for (int i = 2; i <= n; i++) {
        fac[i] = fac[i - 1]*i % mod;
    }
    inv[n] = quick(fac[n], mod - 2);
    for (int i = n-1; i >= 0 ; i--)inv[i] = inv[i+1]*(i+1) % mod;
}

//n大
ll C(int n, int m) {
    return fac[n] * inv[m] % mod * inv[n - m] % mod;
}
```

5.9 快速阶乘

```
//minamoto
#include<bits/stdc++.h>
#define R register
#define ll long long
#define fp(i,a,b) for(R int i=(a),I=(b)+1;i<I;++i)
#define fd(i,a,b) for(R int i=(a),I=(b)-1;i>I;--i)
#define go(u) for(int i=head[u],v=e[i].v;i;i=e[i].nx,v=e[i].v)
using namespace std;
const int N=(1<<17)+5;int P;
inline int add(R int x,R int y){return 0ll+x+y>=P?0ll+x+y-P:x+y;}
inline int dec(R int x,R int y){return x-y<0?x-y+P:x-y;}
inline int mul(R int x,R int y){return 1ll*x*y-1ll*x*y/P*P;}
int ksm(R int x,R int y){
    R int res=1;
    for(;y;y>>=1,x=mul(x,x))(y&1)?res=mul(res,x):0;
    return res;
}
const double Pi=acos(-1.0);
struct cp{
    double x,y;
    inline cp(){}
    inline cp(R double xx,R double yy):x(xx),y(yy){}
    inline cp operator +(const cp &b)const{return cp(x+b.x,y+b.y);}
    inline cp operator -(const cp &b)const{return cp(x-b.x,y-b.y);}
    inline cp operator *(const cp &b)const{return cp(x*b.x-y*b.y,x*b.y+y*b.x);}
    inline cp operator *(const double &b)const{return cp(x*b,y*b);}
    inline cp operator ~()const{return cp(x,-y);}
}w[2][N];
int r[21][N],ifac[N],lg[N],inv[N];double iv[21];
void Pre(){
    iv[0]=1;
    fp(d,1,17){
```

```
33          fp(i,0,(1<<d)-1)r[d][i]=(r[d][i>>1]>>1)|((i&1)<<(d-1));
34          lg[1<<d]=d,iv[d]=iv[d-1]*0.5;
35      }
36      inv[0]=inv[1]=ifac[0]=ifac[1]=1;
37      fp(i,2,131072)inv[i]=mul(P-P/i,inv[P%i]),ifac[i]=mul(ifac[i-1],inv[i]);
38      for(R int i=1,d=0;i<131072;i<<=1,++d)fp(k,0,i-1)
39          w[1][i+k]=cp(cos(Pi*k*iv[d]),sin(Pi*k*iv[d])),
40          w[0][i+k]=cp(cos(Pi*k*iv[d]),-sin(Pi*k*iv[d]));
41  }
42  int lim,d;
43  void FFT(cp *A,int ty){
44      fp(i,0,lim-1)if(i<r[d][i])swap(A[i],A[r[d][i]]);
45      cp t;
46      for(R int mid=1;mid<lim;mid<<=1)
47          for(R int j=0;j<lim;j+=(mid<<1))
48              fp(k,0,mid-1)
49                  A[j+k+mid]=A[j+k]-(t=w[ty][mid+k]*A[j+k+mid]),
50                  A[j+k]=A[j+k]+t;
51      if(!ty)fp(i,0,lim-1)A[i]=A[i]*iv[d];
52  }
53  void MTT(int *a,int *b,int len,int *c){
54      static cp f[N],g[N],p[N],q[N];
55      lim=len,d=lg[lim];
56      fp(i,0,len-1)f[i]=cp(a[i]>>16,a[i]&65535),g[i]=cp(b[i]>>16,b[i]&65535);
57      fp(i,len,lim-1)f[i]=g[i]=cp(0,0);
58      FFT(f,1),FFT(g,1);
59      fp(i,0,lim-1){
60          cp t,f0,f1,g0,g1;
61          t=~f[i?lim-i:0],f0=(f[i]-t)*cp(0,-0.5),f1=(f[i]+t)*0.5;
62          t=~g[i?lim-i:0],g0=(g[i]-t)*cp(0,-0.5),g1=(g[i]+t)*0.5;
63          p[i]=f1*g1,q[i]=f1*g0+f0*g1+f0*g0*cp(0,1);
64      }
65      FFT(p,0),FFT(q,0);
66      fp(i,0,lim-1)c[i]=((((ll)(p[i].x+0.5)%P<<16)%P<<16)+((ll)(q[i].x+0.5)<<16)+((ll)(q[
        i].y+0.5)))%P;
67  }
68  void calc(int *a,int *b,int n,int k){
69      static int f[N],g[N],h[N],sum[N],isum[N];
70      int len=1;while(len<=n+n)len<<=1;
71      fp(i,0,n)f[i]=mul(a[i],mul(ifac[i],ifac[n-i]));
72      for(R int i=n-1;i>=0;i-=2)f[i]=P-f[i];
73      int t=dec(k,n);
74      fp(i,0,n+n)g[i]=add(i,t);
75      sum[0]=g[0];fp(i,1,n+n)sum[i]=mul(sum[i-1],g[i]);
76      isum[n+n]=ksm(sum[n+n],P-2);
77      fd(i,n+n,1)isum[i-1]=mul(isum[i],g[i]);
78      fp(i,1,n+n)g[i]=mul(isum[i],sum[i-1]);g[0]=isum[0];
79      fp(i,n+1,len-1)f[i]=0;fp(i,n+n+1,len-1)g[i]=0;
80
81      MTT(f,g,len,h);
82      int res=1,p1=k-n,p2=k;
83      fp(i,p1,p2)res=1ll*res*i%P;
84      res=dec(res,0);
85
86      fp(i,0,n)g[i]=(0ll+P+p1+i)%P;
87      sum[0]=g[0];fp(i,1,n)sum[i]=mul(sum[i-1],g[i]);
88      isum[n]=ksm(sum[n],P-2);
89      fd(i,n,1)isum[i-1]=mul(isum[i],g[i]);
90      fp(i,1,n)g[i]=mul(isum[i],sum[i-1]);g[0]=isum[0];
```

```
91
92      for(R int i=0;i<=n;p2=add(p2,1),++i)
93          b[i]=mul(h[i+n],res),res=mul(res,mul(g[i],p2+1));
94  }
95  int solve(int bl){
96      static int a[N],b[N],c[N];
97      int s=0;for(int p=bl;p;p>>=1)++s;a[0]=1,--s;
98      int qwq=ksm(bl,P-2);
99      for(int p=0;s>=0;--s){
100         if(p){
101             calc(a,b,p,p+1);
102             fp(i,0,p)a[p+i+1]=b[i];a[p<<1|1]=0;
103             calc(a,b,p<<1,mul(p,qwq));
104             p<<=1;fp(i,0,p)a[i]=mul(a[i],b[i]);
105         }
106         if(bl>>s&1){
107             fp(i,0,p)a[i]=mul(a[i],(1ll*bl*i+p+1)%P);
108             p|=1,a[p]=1;
109             fp(i,1,p)a[p]=mul(a[p],(1ll*bl*p+i)%P);
110         }
111     }
112     int res=1;
113     fp(i,0,bl-1)res=mul(res,a[i]);
114     return res;
115 }
116 int GetFac(int n){
117     int s=sqrt(n),res=solve(s);
118     fp(i,s*s+1,n)res=mul(res,i);
119     return res;
120 }
121 int Fac(int n){
122     if(n>P-1-n){
123         int res=ksm(GetFac(P-1-n),P-2);
124         return (P-1-n)&1?res:P-res;
125     }
126     return GetFac(n);
127 }
128 int n;
129 int main(){
130 //  freopen("testdata.in","r",stdin);
131     scanf("%d%d",&n,&P),Pre();
132     printf("%d\n",Fac(n));
133     return 0;
134 }
```

5.10 FFT

```
1  inline int lowbit(int x) { return x & -x; }
2
3  int calc(int n) {
4      int k = 0;
5      while ((1 << k) < n) k++;
6      return k;
7  }
8
9  // FFT
10 const double pi = acos(-1.0);
11
```

```
const int N = (1 << 20);
using Complex = complex<double>;

void change(Complex p[], int n) {
    int k = calc(n);
    n = 1 << k;
    vector<int> r(n, 0);
    for (int i = 0; i < n; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (k - 1));
    for (int i = 0; i < n; i++) if (i < r[i]) swap(p[i], p[r[i]]);
}

void FFT(Complex p[], int n, int type) {
    change(p, n);
    for (int mid = 1; mid < n; mid <<= 1) { //待合并区间的长度的一半
        Complex wn(cos(pi / mid), type * sin(pi / mid)); //单位根
        for (int R = mid << 1, j = 0; j < n; j += R) { //R是区间的长度，j表示前已经到哪个位置了
            Complex w(1, 0); //幂
            for (int k = 0; k < mid; k++, w = w * wn) { //枚举左半部分
                Complex x = p[j + k], y = w * p[j + mid + k]; //蝴蝶效应
                p[j + k] = x + y;
                p[j + mid + k] = x - y;
            }
        }
    }
}

1.大数相乘
给出两个n位10进制整数x和y，你需要计算x*y。
int n,rev[N];
char x[N],y[N];
Complex a[N],b[N];
int ans[N];

void fft(Complex p[],int n,int type){
    for(int i=0;i<n;i++) if(i<rev[i]) swap(p[i],p[rev[i]]);
    for (int mid = 1; mid < n; mid <<= 1) { //待合并区间的长度的一半
        Complex wn(cos(pi / mid), type * sin(pi / mid)); //单位根
        for (int R = mid << 1, j = 0; j < n; j += R) { //R是区间的长度，j表示前已经到哪个位置了
            Complex w(1, 0); //幂
            for (int k = 0; k < mid; k++, w = w * wn) { //枚举左半部分
                Complex x = p[j + k], y = w * p[j + mid + k]; //蝴蝶效应
                p[j + k] = x + y;
                p[j + mid + k] = x - y;
            }
        }
    }
}

int main() {
    scanf("%d",&n);
    scanf("%s%s",x,y);
    for(int i=n-1;i>=0;i--) a[n-1-i].real(x[i]-'0');
    for(int i=n-1;i>=0;i--) b[n-1-i].real(y[i]-'0');
    int top=1,bit=0;
    while(top<=(n<<1)) top<<=1,bit++;
    for(int i=0;i<top;i++) rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));
    fft(a,top,1); fft(b,top,1);
```

```
    for(int i=0;i<top;i++) a[i]=a[i]*b[i];
    fft(a,top,-1);
    for(int i=0;i<top;i++){
        ans[i]+=(int)(a[i].real()/top+0.5);
        if(ans[i]>=10){
            ans[i+1]+=ans[i]/10; ans[i]%=10; top+=(i==top);
        }
    }
    while(!ans[top]&&top>=1) top--;
    top++;
    while(--top>=0) printf("%d",ans[top]);
}
```

5.11 平面最近点对

```
int n;
int a[N],tot;

struct node{
    double x,y;
}p[N];

bool cmp(node a,node b){
    if(a.x!=b.x) return a.x<b.x;
    return a.y<b.y;
}

bool cmp2(int i,int j){
    return p[i].y<p[j].y;
}

double dist(int i,int j){
    return sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x)+(p[i].y-p[j].y)*(p[i].y-p[j].y));
}

double merge(int left,int right){
    double d = 1e18;
    if(left==right) return d;
    if(left+1==right) return dist(left,right);
    int mid=(left+right)>>1;
    double d1=merge(left,mid),d2=merge(mid+1,right);
    d=min(d1,d2);
    tot=0;
    for(int i=left;i<=right;i++) if(fabs(p[mid].x-p[i].x)<d) a[++tot]=i;
    sort(a+1,a+1+tot,cmp2);
    for(int i=1;i<=tot;i++){
        for(int j=i+1;j<=tot&&p[a[j]].y-p[a[j]].y<d;j++){
            double d3=dist(a[i],a[j]);
            d=min(d,d3);
        }
    }
    return d;
}

int main() {
#ifdef ACM_LOCAL
    freopen("./std.in", "r", stdin);
#endif
```

```
44      scanf("%d",&n);
45      for(int i=1;i<=n;i++) scanf("%lf%lf",&p[i].x,&p[i].y);
46      sort(p+1,p+1+n,cmp);
47      printf("%.4lf\n",merge(1,n));
48  }
```

6 杂七杂八

6.1 二分、三分查找

```
// 查找第一个相等的元素
int findFirstEqual(int[] array, int key) {
    int left = 0;
    int right = array.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (array[mid] >= key)
            right = mid - 1;
        else
            left = mid + 1;
    }
    if (left < array.length && array[left] == key)
        return left;
    return -1;
}

// 查找最后一个相等的元素
int findLastEqual(int[] array, int key) {
    int left = 0;
    int right = array.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (array[mid] <= key)
            left = mid + 1;
        else
            right = mid - 1;
    }
    if (right >= 0 && array[right] == key)
        return right;
    return -1;
}

浮点查找最后一个相等的元素，因为浮点区间连续，所以不需要整数一样+-1
double left=0,right=2000;
while(left+eps<=right){
      double mid = (left+right)/2;
      if(isOK(mid)) left=mid;
      else right=mid;
}
printf("%lld\n",(ll)(1000*right));

// 查找最后一个等于或者小于key的元素
int findLastEqualSmaller(int[] array, int key) {
    int left = 0;
    int right = array.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (array[mid] > key)
            right = mid - 1;
        else
            left = mid + 1;
    }
    return right;
}

```

```
// 查找最后一个小于key的元素
int findLastSmaller(int[] array, int key) {
    int left = 0;
    int right = array.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (array[mid] >= key)
            right = mid - 1;
        else
            left = mid + 1;
    }
    return right;
}

// 查找第一个等于或者大于key的元素
int findFirstEqualLarger(int[] array, int key) {
    int left = 0;
    int right = array.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (array[mid] >= key)
            right = mid - 1;
        else
            left = mid + 1;
    }
    return left;
}

// 查找第一个大于key的元素
int findFirstLarger(int[] array, int key) {
    int left = 0;
    int right = array.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (array[mid] > key)
            right = mid - 1;
        else
            left = mid + 1;
    }
    return left;
}


当二分的函数值不是递增/减，而是先增后减或者先减后增时二分就挂了。此时使用三分，需要注意的是必须严格递增或递减

//当存在极小值时
while(left+eps<=right){
    double lm=left+(right-left)/3;
    double rm=right-(right-left)/3;
    if(calc(lm)>=calc(rm)) left=lm;
    else right=rm;
}
printf("%.10lf\n",calc(left));

//当存在极大值时
while(left+eps<=right){
    double lm=left+(right-left)/3;
    double rm=right-(right-left)/3;
```

```
    if(calc(lm)>=calc(rm)) right=rm;
    else left=lm;
}
printf("%.10lf\n",calc(left));
```

6.2 离散化

```
1、离散化
for (int i = 1; i <= n; i++) {
    scanf("%d", &a[i]);
    b[i] = a[i];
}
sort(b + 1, b + 1 + n);
m = unique(b + 1, b + 1 + n) - b - 1;
for (int i = 1; i <= n; i++) {
    a[i] = lower_bound(b + 1, b + 1 + m, a[i]) - b;
}
```

6.3 斯坦纳树

```
给定n个点,m条边，请选择一些边，使得1<=i<=d(1<=d<=4),i号节点和n - i + 1号节点可以通过选中的边连通，
    最小化选中的所有边的权值和。
首先我们设计状态：f[i][j]表示根为i，连通状态为j的最小代价（状态只李记录关键点）
有两种转移方法：
枚举子树的形态：f[i][j]=min(f[i][j]，f[i][k]+f[i][l])，其中k和l是对j的一个划分
按照边进行松弛：f[i][j]=min(f[i][j]，f[i’][j]+w[i][i’])，其中i和i’之间有边相连
对于第一种转移，我们直接枚举子集
对于第二种转移，我们用spfa进行状态转移

int n,m,d;

struct node{
    int to,v,nx;
}edge[N<<1];
int tot,head[N];

void add(int from,int to,int v) {
    edge[tot].to=to;
    edge[tot].v=v;
    edge[tot].nx=head[from];
    head[from]=tot++;
}

class SteinerTree{
public:
    int f[260][N],g[260];
    queue<int>q;
    bool in[N];

    void spfa(int S) {
        while (!q.empty()) {
            int from=q.front(); q.pop();
            in[from]=0;
            for (int i=head[from];i;i=edge[i].nx) {
                int to=edge[i].to;
                if (f[S][to]>f[S][from]+edge[i].v) {
                    f[S][to]=f[S][from]+edge[i].v;
```

```
                    if (!in[to]) {
                        in[to]=1;
                        q.push(to);
                    }
                }
            }
        }

    void solve(){
        memset(f,0x3f,sizeof(f));
        int cnt=2*d;
        for (int i=1;i<=d;i++) f[1<<i-1][i]=0,f[1<<d+i-1][n-i+1]=0;
        int top=(1<<cnt);
        for (int S=1;S<top;S++) {
            for (int s=(S-1)&S;s;s=(s-1)&S) {
                int t=S^s;
                for (int i=1;i<=n;i++)
                    f[S][i]=min(f[S][i],f[s][i]+f[t][i]);
            }
            for (int i=1;i<=n;i++)
                if (f[S][i]<INF&&!in[i])
                    q.push(i),in[i]=1;
            spfa(S);
        }
        memset(g,0x3f,sizeof(g));
        top=(1<<d);
        for (int S=1;S<top;S++)
            for (int i=1;i<=n;i++)
                g[S]=min(g[S],f[S^(S<<d)][i]);
        for (int S=1;S<top;S++)
            for (int s=(S-1)&S;s;s=(s-1)&S)
                g[S]=min(g[S],g[s]+g[S^s]);
        printf("%d",g[top-1]==INF? -1:g[top-1]);
    }
}steinerTree;

int main(){
    int cin_x,cin_y,cin_v;
    scanf("%d%d%d",&n,&m,&d);
    tot=1;
    for(int i=1;i<=m;i++){
        scanf("%d%d%d",&cin_x,&cin_y,&cin_v);
        add(cin_x,cin_y,cin_v);
        add(cin_y,cin_x,cin_v);
    }
    steinerTree.solve();
}
```

6.4 子矩阵问题

```
1、01矩阵求第二大全是1矩阵，单调栈
int main() {
    int hi, li, top;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%s", s[i] + 1);
    for (int i = 1; i <= n; i++) {
```

```
        for (int j = 1; j <= m; j++) h[i][j] = (s[i][j] == '1') ? h[i - 1][j] + 1 : 0;
        h[i][m + 1] = 0; st[top = 0] = 0;
        for (int j = 1; j <= m + 1; j++) {
            if (h[i][j] == h[i][st[top]]) st[top] = j;
            else if (h[i][j] > h[i][st[top]]) st[++top] = j;
            else {
                int t = st[top];
                do {
                    hi = h[i][st[top]]; li = (t - st[--top]); //求长度和高度
                    ans.push_back(hi * li); //计算面积
                    ans.push_back(max((hi - 1) * li, hi * (li - 1)));
                } while (h[i][j] < h[i][st[top]]);
                if (h[i][j] > h[i][st[top]]) top++;
                st[top] = j;
            }
        }
        sort(ans.begin(), ans.end(), greater<int>());
        ans.resize(2);
    }
    printf("%d\n", ans[1]);
}

2、01矩阵求最大全是1矩阵，如需求第二大子矩阵则需要将int a=i,b=r[i][j],R=r[i][j]-l[i][j]+1,H=h[i
    ][j]；四点进行去重即可
void solve(){
    int maxl,maxr;
    for(int i=1;i<=m;i++) r[0][i]=m;
    for(int i=1;i<=n;i++){
        maxl=1; maxr=m;
        for(int j=1;j<=m;j++){
            if(s[i][j]=='0'){
                maxl=j+1;
                h[i][j]=l[i][j]=0;
            }else{
                h[i][j]=h[i-1][j]+1;
                l[i][j]=max(maxl,l[i-1][j]);
            }
        }
        for(int j=m;j>=1;j--){
            if(s[i][j]=='0'){
                maxr=j-1;
                r[i][j]=m;
            }else{
                r[i][j]=min(maxr,r[i-1][j]);
                int R=r[i][j]-l[i][j]+1,H=h[i][j];
                ans=max(ans,R*H);
            }
        }
    }
    printf("%d\n",ans);
}

//简单悬线法
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= m; j++)
    scanf(" %c", s[i] + j), lft[i][j] = rgt[i][j] = j, up[i][j] = 1;
for (int i = 1; i <= n; i++) {
  for (int j = 1; j <= m; j++)
    if (s[i][j] == '0' && s[i][j - 1] == '0') lft[i][j] = lft[i][j - 1];
```

```
  for (int j = m; j >= 1; j--)
    if (s[i][j] == '0' && s[i][j + 1] == '0') rgt[i][j] = rgt[i][j + 1];
}
for (int i = 1; i <= n; i++)
  for (int j = 1; j <= m; j++) {
    if (i > 1 && s[i][j] == '0' && s[i - 1][j] == '0') {
      lft[i][j] = max(lft[i][j], lft[i - 1][j]);
      rgt[i][j] = min(rgt[i][j], rgt[i - 1][j]);
      up[i][j] = up[i - 1][j] + 1;
    }
    ans = max(ans, (rgt[i][j] - lft[i][j] + 1) * up[i][j]);
}

3、优先队列+并查集求第二大1矩阵
int find(int x){
    if (par[x] == x)return x;
    return par[x] = find(par[x]);
}

void merge(int x,int y){
    int fx=find(x),fy=find(y);
    par[fy]=fx;
    len[fx]+=len[fy];
    return;
}

int main() {
    scanf("%d%d",&n,&m);
    int ans=0,ans2=0;
    for(int i=1;i<=n;i++){
        scanf("%s",s+1);
        for(int j=1;j<=m;j++){
            if(s[j]=='0') h[j]=0;
            else{
                h[j]++;
                q.push(node(j,h[j]));
            }
            len[j]=1;par[j]=j;vis[j]=0;
        }
        while(!q.empty()){
            node t=q.top(); q.pop();
            int pos=t.pos,hi=t.height;
            vis[pos]=1;
            if(vis[pos-1]) merge(pos-1,pos);
            if(vis[pos+1]) merge(pos+1,pos);
            int f=find(pos);
            if(len[f]*hi>=ans) ans2=ans,ans=len[f]*hi;
            else ans2=max(ans2,len[f]*hi);
            ans2=max(ans2,max((len[f]-1)*hi,len[f]*(hi-1)));
        }
    }
    printf("%d\n",ans2);
}

4、给定一个矩阵n*m(1<=n,m<=500),求其中最大子矩阵满足矩阵中|最大值-最小值|<=K的大小
void solve(){
    int ans=0,ans2=0;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++) { mx[j]=-inf; mn[j]=inf;}
```

```
        for(int j=i;j<=n;j++){
            for(int k=1;k<=m;k++) mx[k]=max(mx[k],a[j][k]);
            for(int k=1;k<=m;k++) mn[k]=min(mn[k],a[j][k]);
            int l=1,head=1,tail=0,head2=1,tail2=0;
            for(int r=1;r<=m;r++){
                while(head<=tail&&mx[r]>=mx[q[0][tail]]) tail--;
                while(head2<=tail2&&mn[r]<=mn[q[1][tail2]]) tail2--;
                q[0][++tail]=r;
                q[1][++tail2]=r;
                while(l<=r&&mx[q[0][head]]-mn[q[1][head2]]>k){
                    l++;
                    if(q[0][head]<l) head++;
                    if(q[1][head2]<l) head2++;
                }
                ans=max(ans,(j-i+1)*(r-l+1)); //求最大子矩阵
                ans+=r-l+1; //求所有满足条件的子矩阵
            }
        }
    }
    printf("%d\n",ans);
}
```

6.5 矩阵模板

```
1、
friend bool operator< (node a,node b){
        return a.step>b.step;
}
优先队列将步数较小的放在前面

2、
do {
    if (check()) {
    ......
    }
} while (next_permutation(a + 1, a + 1 + n));
全排列函数，注意最初需要将a从小到大进行排序。


矩阵模板:
mat Mul(mat a, mat b) {
    mat c;
    for (int i = 0; i < ac.tot; i++) {
        for (int j = 0; j < ac.tot; j++) {
            ull sum = 0;
            for (int k = 0; k < ac.tot; k++)
                sum = sum + a.arr[i][k] * b.arr[k][j];
            c.arr[i][j] = sum;
        }
    }
    return c;
}

mat quick(mat a, int b) {
    mat res = ones;
    while (b) {
        if (b & 1)
            res = Mul(res, a);
```

```
        b >>= 1;
        a = Mul(a, a);
    }
    return res;
}

mat add(mat a, mat b) {
    mat c;
    for (int i = 0; i < ac.tot; i++) {
        for (int j = 0; j < ac.tot; j++) {
            c.arr[i][j] = a.arr[i][j] + b.arr[i][j];
        }
    }
    return c;
}

求: A^1+A^2+A^3+A^4...+A^K
mat getsum(int k) {
    if (k == 1)
        return A;
    mat t = getsum(k / 2);
    if (k & 1) {
        mat reminder = quick(A, k);
        mat cur = quick(A, (k - 1) / 2);
        t = add(t, Mul(cur, t));
        t = add(reminder, t);
    }
    else {
        mat cur = quick(A, k / 2);
        t = add(t, Mul(cur, t));
    }
    return t;
}
```

6.6 汉诺塔

```
汉诺塔:
    其实算法非常简单，当盘子的个数为n时，移动的次数应等于2^n-1。只要轮流进行两步操作就可以了。首先把三根
柱子按顺序排成品字型，把所有的圆盘按从大到小的顺序放在柱子A上，根据圆盘的数量确定柱子的排放顺序：若n为偶数，
按顺时针方向依次摆放 A B C；若n为奇数，按顺时针方向依次摆放 A C B。
    □ 按顺时针方向把圆盘1从现在的柱子移动到下一根柱子，即当n为偶数时，若圆盘1在柱子A，则把它移动到B；若圆
盘1在柱子B，则把它移动到C；若圆盘1在柱子C，则把它移动到A。
    □ 接着，把另外两根柱子上可以移动的圆盘移动到新的柱子上。即把非空柱子上的圆盘移动到空柱子上，当两根柱子
都非空时，移动较大的圆盘。这一步没有明确规定移动哪个圆盘，你可能以为会有多种可能性，其实不然，可实施的行动是唯一的。
    □ 反复进行□ □ 操作，最后就能按规定完成汉诺塔的移动。
    所以结果非常简单，就是按照移动规则向一个方向移动金片:
    如3阶汉诺塔的移动: A→C,A→B,C→B,A→C,B→A,B→C,A→C

void Move(int n, char a, char b){
    printf("第%d次移动 Move %d: Move from %c to %c !\n",++cnt,n,a+'A',b+'A');
}

void Hanoi(int n,int a,int b,int c){
```

```
    if(n==1){
        Move(n,a,c);
    }else{
        Hanoi(n - 1, a, c, b);
        Move(n, a, c);
        Hanoi(n - 1, b, a, c);
    }
}


求汉诺塔具体操作数结果:
int main(){
    int ab=0,ac=1,ba=2,bc=3,ca=4,cb=5;
    scanf("%d",&n);
    ret[ac]++; dp[1][ac]=1;
    for(int i=2;i<=n;i++){
        if(i%2==0){
            dp[i][ab]+=dp[i-1][ac]+dp[i-1][cb];
            dp[i][bc]+=dp[i-1][ac]+dp[i-1][ba];
            dp[i][ca]+=dp[i-1][ba]+dp[i-1][cb];
        }else{
            dp[i][ac]+=dp[i-1][ab]+dp[i-1][bc];
            dp[i][ba]+=dp[i-1][bc]+dp[i-1][ca];
            dp[i][cb]+=dp[i-1][ab]+dp[i-1][ca];
        }
        for(int j=0;j<6;j++) ret[j]+=dp[i][j];
    }
    printf("A->B:%lld\n", ret[0]);
    printf("A->C:%lld\n", ret[1]);
    printf("B->A:%lld\n", ret[2]);
    printf("B->C:%lld\n", ret[3]);
    printf("C->A:%lld\n", ret[4]);
    printf("C->B:%lld\n", ret[5]);
    printf("SUM:%lld\n", pw(n)-1);
}
```

6.7 前缀和与差分

```
1、前缀和与差分
    对于一个数组a定义数组s[i]=sigma(j=1,i)a[j]
//为了避免数组越位，下标从1开始用
for(int i=1;i<=n;i++)
    s[i]=s[i-1]+a[i];

    定义数组d[i]= 1、i==0,di[i]=a[i] 2、i>=1,d[i]=a[i]-a[i-1]

//为了避免数组越位，下标从1开始用
for(int i=n;i>=1;i--)
    d[i]=a[i]-a[i-1];

    发现对于原数组a的区间加数操作对应差分数组d只改变了两个地方。因为差分数组的前缀和数组为原数组，所以对差分数组的
修改，在原数组上产生的影响是这个位置以后的一个后缀影响。给d[l]加上x就相当于给a[l],a[l+1],a[l+2]....a[n]全部加上x。
给d[r+1]加上-x就相当于给a[r+1],r[r+2],....a[n]全部加上-x。那么如果要给a[l],a[l+1]...a[r]全部加上x就很简单了。

void add(int l,int r,int x){
```

```
    d[l]+=x;
    d[r+1]-=x;
}

注意我们操作的是数组d，是差分数组，不是原数组。也就是说如果你最后要输出原数组a的话还要在做一遍前缀和还原。

2、静态维护区间加等差数列的求和问题
    维护一个数组，先进行m次操作，然后查询每个位置的值，每个操作给定四个参数l,r,a,k表示从l到r依次加上一个首项为a，公差为k的等差数列。
其中[l,r]区间分别加上[a,a+k,a+2k,...a+(r-l)k]

int n,m,d2[N],l,r,a,k;
void add(int l,int r,int a,int k){
    d2[l]+=a;
    d2[l+1]+=k-a;
    d2[r+1]-=(r-l+1)*k+a;
    d2[r+2]-=(l-r)*k-a;
}
void iter(){
    for(int i=1;i<=n;++i) d2[i]+=d2[i-1];
}

int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;++i){
        scanf("%d%d%d%d",&l,&r,&a,&k);
        add(l,r,a,k);
    }
    iter(); //第一次为了将每项额外加的补齐
    iter(); //还原完整数组
    for(int i=1;i<=n;++i) printf("%d%c",d2[i],i==n?'\n':' ');
    return 0;
}

3、二维前缀和与差分
    对于一个二维数组a定义s[i][j]=sigma(p=0,i)sigma(q=0,j)a[p][q]为数组a的前缀和数组
//为了避免数组越位，下标从1开始
for(int i=1;i<=n;i++){
    for(int j=1;j<=m;j++){
        s[i][j]=s[i-1][j]+s[i][j-1]-s[i-1][j-1]+a[i][j];
    }
}
    那么定义d[i][j]为差分数组
为了避免数组越位，下标从1开始
for(int i=n;i;i--){
    for(int j=m;j;j--){
        d[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1];
    }
}

静态数组的求和问题
sum(l1,r1,l2,r2)=s[r1][r2]-s[l1-1][r2]-s[r1][l2-1]+s[l1-1][l2-1]。这个是利用了简单的容斥原理，纸上画画图就能理解
因为s[i][j]是一个左上矩形的矩阵和，所以s[r1][r2]这个矩阵在减去s[l1-1][r2]与s[r1][l2-1]矩阵后，它们共有的s[l1-1][l2-1]
部分被减了两次，所以再加上一次。
ll sum(int l1,int r1,int l2,int r2){
    return s[r1][r2]-s[l1-1][r2]-s[r1][l2-1]+s[l1-1][l2-1];
```

```
}

进行m次区间修改后的静态单点求值问题
    推导过程类似一维的前缀和与差分，其实就是反过来考虑差分数组对原数组的影响是一个后缀影响（这里可以理解为影响整个右下角矩阵）
void add(int l1,int r1,int l2,int r2,int x){
    d[l1][l2]+=x;
    d[r1+1][l2]-=x;
    d[l1][r2+1]-=x;
    d[r1+1][r2+1]+=x;
}
    同理，d数组可以靠一次前缀和操作还原为原数组，如果是矩阵求和问题还可再做一遍前缀和。


4、高维前缀和
    同理三维只需要三个维度进行处理即可
for(int i=1;i<=n;++i){
    for(int j=1;j<=m;++j){
        a[i][j]=a[i][j]+a[i][j-1];
    }
}
for(int i=1;i<=n;++i){
    for(int j=1;j<=m;++j){
        a[i][j]=a[i][j]+a[i-1][j];
    }
}

5、状压dp前缀和
for(int i=0;i<w;++i)//依次枚举每个维度
{
    for(int j=0;j<(1<<w);++j)//求每个维度的前缀和
    {
        if(j&(1<<i))s[j]+=s[j^(1<<i)];
    }
}


6、菱形差分
题目:
    地方阵地可以看做是n×m的矩形，航空母舰总共会派出q架飞机。飞机有两种，第一种飞机会轰炸以(xi，yi)为中心，
对角线长为li的正菱形(也就是两条对角线分别于x轴 y轴平行的正方形)，而第二种飞机只会轰炸正菱形的上半部分(包括第xi行)
(具体看样例解释)。现在小a想知道所有格子被轰炸次数的异或和，注意：不保证被轰炸的格子一定在矩形范围内，若越界请忽略

输入:
    第一行三个整数n，m，q分别表示矩阵的长/宽/询问次数，接下来q行，每行四个整数opt,x,y,l表示飞机类型，轰炸的坐标，以及对角线长度
保证l为奇数!

4 5 4
1 2 2 1
1 3 3 5
1 3 2 3
2 2 4 3

轰炸后结果为:

```

```
0 0 1 1 0
0 3 2 2 1
2 2 2 1 1
0 2 1 1 0
最后把所有元素异或后为2

代码:
int n,m,q,base=500;
int op,x,y,L;
int a[N][N],b[N][N];

void down(int x,int y,int L){
    a[x+L+1][y]++;
    a[x+1][y+L]--;

    b[x+1][y-L+1]++;
    b[x+L+1][y+1]--;
}

void up(int x,int y,int L){
    a[x-L][y]++;
    a[x+1][y-L-1]--;

    b[x-L][y+1]--;
    b[x+1][y+2+L]++;
}

int main() {
    read(n); read(m); read(q);
    while(q--){
        read(op); read(x); read(y); read(L);
        x+=base; y+=base;
        if(op==1) down(x,y,L/2);
        up(x,y,L/2);
    }
    int ans=0;
    for(int i=1;i<=n+base*2;i++){
        int x=0;
        for(int j=1;j<=m+base*2;j++){
            x+=a[i][j]+b[i][j];
            if(i>base&&i<=n+base&&j>base&&j<=m+base) ans^=x;
            a[i+1][j-1]+=a[i][j];
            b[i+1][j+1]+=b[i][j];
        }
    }
    printf("%d\n",ans);
}
```

6.8 stl

```
1、lower_bound,upper_bound
    lower_bound()和upper_bound()都是利用二分查找的方法在一个排好序的数组中进行查找的。

    在从小到大的排序数组中:
    1、lower_bound(begin,end,num): 从数组的begin位置到end-1位置二分查找第一个大于或等于num的数字，找到返回该数
字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。
```

```
    2、upper_bound( begin,end,num): 从数组的begin位置到end-1位置二分查找第一个大于num的数字，找到返回该数字的
地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

    在从大到小的排序数组中，重载lower_bound()和upper_bound():
    1、lower_bound( begin,end,num,greater<type>()):从数组的begin位置到end-1位置二分查找第一个小于或等于num
的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。
    2、upper_bound( begin,end,num,greater<type>() ):从数组的begin位置到end-1位置二分查找第一个小于num的数字，
找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。


//数组测试
int a[N];

int main(){
    int n=10;
    //从小到大,lower_bound测试,找第一个大于或等于num的数字，超出去一律返回位置n
    for(int i=0;i<n;i++) a[i]=i;
    int i=lower_bound(a,a+n,n+n)-a;
    printf("%d %d\n",i,a[i]);
    //从小到大,upper_bound测试,第一个大于num的数字，超出去一律返回位置n
    i=upper_bound(a,a+n,100)-a;
    printf("%d %d\n",i,a[i]);
    //从大到小,lower_bound测试,找第一个小于或等于num的数字，若小于所有值，则一律返回n
    for(int i=0;i<n;i++) a[i]=n-i-1;
    i=lower_bound(a,a+n,-1,greater<int>())-a;
    printf("%d %d\n",i,a[i]);
    //从大到小,upper_bound测试,第一个小于num的数字，若小于所有值，则一律返回n
    i=upper_bound(a,a+n,-999,greater<int>())-a;
    printf("%d %d\n",i,a[i]);

    //情况上面情况一样
    vector<int>v;
    for(int i=0;i<n;i++) v.push_back(i);
    int i=lower_bound(v.begin(),v.end(),23)-v.begin();
    printf("%d %d\n",i,v[i]);
}


2、结构体中比较函数作用于数组和set
//没有t的是i，t是j,i<j，因此返回x较小，如果x一样大，返回y大的
struct node{
    int x,y;
    bool operator <(const node&t)const{
        if(x==t.x) return y>t.y;
        return x<t.x;
    }
};

int main(){
#ifdef ACM_LOCAL
    freopen("./std.in", "r", stdin);
    //freopen("./std.out","w",stdout);
#endif
    set<node>st;
    st.insert(node{1,1});
```

```
    st.insert(node{1,3});
    st.insert(node{2,2});
    node p[N];
    p[1]=node{1,1}; p[2]=node{1,3}; p[3]=node{2,2};
    sort(p+1,p+1+3);
    //(1,3)(1,1)(2,2)
}

3、优先队列的比较函数
//跟上述情况相反
struct node{
    int x,y;
    bool operator <(const node&t)const{
        if(x==t.x) return y>t.y;
        return x<t.x;
    }
};

int main(){
#ifdef ACM_LOCAL
    freopen("./std.in", "r", stdin);
    //freopen("./std.out","w",stdout);
#endif
    priority_queue<node>q;
    q.push(node{1,3});
    q.push(node{1,1});
    q.push(node{2,1});
    while(!q.empty()){
        node t=q.top(); q.pop();
        printf("%d %d\n",t.x,t.y);
    }
    //(2,1)(1,1)(1,3)
    set<int>st;
    for(int i=1;i<=5;i++) st.insert(i);
    cout<<*(--st.end())<<endl;
    //5
}
```

6.9 最长上升子序列

```
问题1:
    现在想修路，连接两个城市，城市标号分别为x,y。想要路不交叉，求最多能修多少条路
    注意直接求LIS可以得到最优解的个数，但是st里面存储的不是最优解

int main(){
    int Case=1;
    while(scanf("%d",&n)==1){
        for(int i=1;i<=n;i++) scanf("%d%d",&a[i].x,&a[i].y);
        sort(a+1,a+1+n); //x<t.x
        set<int>st;
        st.insert(a[1].y);
        for(int i=2;i<=n;i++){
            auto it=st.lower_bound(a[i].y);
            if(it==st.end()){
                st.insert(a[i].y);
            }else{
                st.erase(it);
                st.insert(a[i].y);
```

```
            }
        }
        printf("Case %d:\n",Case++);
        if(st.size()==1){
            printf("My king, at most %d road can be built.\n\n",st.size());
        }else{
            printf("My king, at most %d roads can be built.\n\n",st.size());
        }
    }
    return 0;
}

问题2:
    现在你有N块矩形木板，第i块木板的尺寸是Xi*Yi，第i块木板能放在第j块木板上方当且仅当Xi<Xj且Yi<Yj，于是你很可能没法
把所有的木板按照一定的次序叠放起来。你想把这些木板分为尽可能少的组，使得每组内的木板都能按照一定的次序叠放。你需要给出任
意一种合理的分组方案。

    根据Dilworth定理，最小组数等于Zi的最长下降子序列长度。因此可以求最长下降子序列，获得结果，并可以得到分组情况

int n;
struct node{
    int x,y,id;
}p[N];

bool cmp(node s,node t){
    if(s.x==t.x) return s.y>t.y;
    return s.x>t.x;
}

int bcc[N];
set<pii>st;

int main(){
#ifdef ACM_LOCAL
    freopen("./std.in", "r", stdin);
    //freopen("./std.out","w",stdout);
#endif
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d%d",&p[i].x,&p[i].y);
        p[i].id=i;
    }
    sort(p+1,p+1+n,cmp);
    int tot=1;
    for(int i=1;i<=n;i++){
        auto it=st.lower_bound(make_pair(p[i].y,0));
        if(it==st.end()){
            bcc[p[i].id]=tot;
            st.insert(make_pair(p[i].y,tot));
            tot++;
        }else{
            bcc[p[i].id]=(*it).second;
            st.erase(it);
            st.insert(make_pair(p[i].y,bcc[p[i].id]));
        }
    }
```

```
75      printf("%d\n",st.size()); //有多少组解
76      for(int i=1;i<=n;i++) printf("%d%c",bcc[i],i==n?'\n':' ');
77  }
```

7 专题训练

7.1 区间 dp

区间dp:顾名思义,区间dp就是在区间上进行动态规划，求解一段区间上的最优解。主要是通过合并小区间的最优解进而得出整个大区间上最优解的dp算法。
其基础代码为:

```
for(int len=2;len<=n;len++){
  for(int i=1;i+len-1<=n;i++){
    int j = i+len-1;
    for(int k=i;k<j;k++){
      dp[i][j]=max(dp[i][j],dp[i][k]+dp[k+1][j]+cost[]);
    }
  }
}
```

其实是一个相对非常暴力的方法去求解最优问题，所以数据量在[100,800]之间的可以考虑O(n^3)求解,而破千通过题目内在关系降到O(n^2)进行求解。而
dp[i][j]的初始化可以在循环内进行，可以通过内外关系一同确定一个区间的最优值。

下面对一些精彩的题目进行分析:

1) Cake(zoj 3537)
题意:
有一块多边形蛋糕，切一刀的代价为|x1+x2|*|y1+y2|%p,如果蛋糕为凸多边形则求出全部切成三角形的代价，否则输出无法切。

思路:
首先使用凸包判断是否为多边形，然后通过凸包得到点的序列，将n->2n,减去环的麻烦。设dp[i][j]表示点[i,j]所组成的多边形被切成三角形所需的
最少代价。可以设k为[i+1,j-1]，通过区间扫描即可获得最终结果。

2) Coloring Brackets(https://codeforces.com/problemset/problem/149/D)
题意:
有一串包含()的字符串，保证合法，最初颜色为黑色，现需要满足3个条件：1.每种括弧都要有一种颜色 2.相匹配的括弧只有一种染了红色或者黑色
3.若括弧被染色，则相邻的括弧颜色不能相同，求最大组合数量，结果mod(1e9+7)。

思路:
本题思路较为奇特,设dp[N][N][3][3]，dp[i][j][x][y]表示在区间[i,j]中，最左边括弧颜色为x,最右边括弧颜色为y的合法括弧序列可以染色的
最大数量。那么接下来只需要分类讨论即可获得最优解。

代码:

```
for(int L=3;L<=len;L++){
        for(int i=1;i+L-1<=len;i++){
            int j = i+L-1;
            if(s[i]=='('&&to[i]==j){
                dp[i][j][0][1] = (dp[i][j][0][1]+dp[i+1][j-1][1][0]+dp[i+1][j-1][0][2]
                +dp[i+1][j-1][2][0]+dp[i+1][j-1][0][0]+dp[i+1][j-1][2][2]+dp[i+1][j
    -1][1][2])%mod;
                dp[i][j][1][0] = (dp[i][j][1][0]+dp[i+1][j-1][0][1]+dp[i+1][j-1][2][0]
                +dp[i+1][j-1][0][2]+dp[i+1][j-1][0][0]+dp[i+1][j-1][2][2]+dp[i+1][j
    -1][2][1])%mod;
                dp[i][j][2][0] = (dp[i][j][2][0]+dp[i+1][j-1][0][1]+dp[i+1][j-1][1][0]
                +dp[i+1][j-1][0][2]+dp[i+1][j-1][0][0]+dp[i+1][j-1][1][1]+dp[i+1][j
    -1][1][2])%mod;
                dp[i][j][0][2] = (dp[i][j][0][2]+dp[i+1][j-1][1][0]+dp[i+1][j-1][0][1]
                +dp[i+1][j-1][2][0]+dp[i+1][j-1][0][0]+dp[i+1][j-1][1][1]+dp[i+1][j
    -1][2][1])%mod;
```

```
            }else if(s[i]=='('&&s[j]==')'){
                int k = to[i];
                for(int x=0;x<3;x++){
                    for(int y=0;y<3;y++){
                        for(int q1=0;q1<3;q1++){
                            for(int q2=0;q2<3;q2++){
                                if(q1==q2&&q1!=0)
                                    continue;
                                dp[i][j][x][y]=(dp[i][j][x][y]+dp[i][k][x][q1]*dp[k+1][
    j][q2][y])%mod;
                            }
                        }
                    }
                }
            }
        }
    }
```

反思:

区间dp不可以局限于简单的二维，可以通过实际情况进行设立情况，本题的难点在于意义的定义，将其定义成合法尤为重要。

3) You Are the One(hdu 4283)

题意:

有一个序列，每个人带着屌丝值di,第k个上场的人会增加评委(k-1)*di，现在有一个小黑屋，可以让人先进去后出来，模拟堆栈，求最少的屌丝值
总和。

思路:

设dp[i][j]表示在[i,j]区间所增加的最少屌丝值，这时候赋予k表示第i个人是第k个人进人，这时候存在 dp[i][j] = min(dp[i][j],
dp[i+1][i+k-1]+dp[i+k][j]+(k-1)*a[i]+k*(pre[j]-pre[i+k-1]))的状态转移方程。

反思:

赋予k特殊含义，来求解。

4) Palindrome subsequence(hdu 4632)

题意:

给一串字符，求解其中有多少个回文串，只要下标不同，回文串之间就是不同。

反思:

设数组为dp[N][N],其中dp[i][j]表示[i,j]区间拥有回文串的最大数量，显然有dp[i][j] = ((dp[i+1][j]+dp[i][j-1]-dp[i+1][j-1]
)%mod+mod)%mod 和 if(s[i-1]==s[j-1]) dp[i][j]=(dp[i][j]+1+dp[i+1][j-1])%mod;

5) Two Rabbits(hdu 4745)

题意:

现有一串石碓，围城一圈，两个兔子分别逆时针和顺时针跳，每次跳的石头权值相同，问最长的可行石碓序列长为多少。

思路:

考虑到一个回文串或者两个回文串可以组成一个满足要求的序列。因此可以先将石碓的回文情况求出然后进行求解。

代码:

```
for(int i=1;i<=n;i++)
    dp[i][i]=1;
for(int len = 2;len<=n;len++){
   for(int i = 1;i+len-1<=2*n;i++){
       int j = i+len-1;
```

```
        dp[i][j]=max(dp[i+1][j],dp[i][j-1]);
        if(a[i]==a[j])
           dp[i][j]=max(dp[i][j],dp[i+1][j-1]+2);
    }
}
int res=1;
for(int i=1;i<=n;i++){
    int j = i+n-1;
    for(int k=i;k<j;k++){
        res = max(res,dp[i][k]+dp[k+1][j]);
    }
}
printf("%d\n",res);
```

6) Sit sit sit (hdu 5151)

题意:

现在有一个被染色为01的椅子，需要安排座位上座顺序，满足三种情况的不能坐：1.两边都有人坐了 2.两边都有椅子 3.左右被坐颜色不同

现求所有可行方案mod(1e9+7)

思路:

设第k个人为最后上座，加上一些组合数知识即可写出。

7) D-game

题意:

现在给了一个序列，并一个D{}集合，序列的公差要满足D集合中元素。1. 在当前剩下的有序数组中选择X(X≥2)个连续数字；2. 检查1选择的X个数字是否构成等差数列，且公差 d∈ {D}；3. 如果2满足，可以在数组中删除这X个数字；4. 重复 1–3 步，直到无法删除更多数字。求最多能删除几个数。

代码:

```
for(int len=2;len<=n;len++){
   for(int i=1;i+len-1<=n;i++){
       int j = i+len-1;
       if(len==2){
          if(mp[a[j]-a[i]])
             dp[i][j]=1;
        }else if(len==3){
             if(mp[a[j-1]-a[i]]&&2*a[j-1]==a[j]+a[i])
                  dp[i][j]=1;
        }else{
             if(mp[a[j]-a[i]]&&dp[i+1][j-1])
                dp[i][j]=1;
             for(int k=i+1;k<j;k++){
                 dp[i][j] |= dp[i][k]&dp[k+1][j];
             }
        }
   }
}
memset(res,0, sizeof(res));
for(int j=1;j<=n;j++){
    res[j]=res[j-1];
    for(int i=1;i<=j;i++){
        if(dp[i][j])
           res[j]=max(res[j],res[i-1]+j-i+1);
    }
}
printf("%d\n",res[n]);
```

7.2 一般 dp

```
1.Photo Processing(cf 883I)
题意:
    给你n个数，将其进行分组，每组最少k个数，求分组方案中最小的最大值差值，其中差值为一组中max(max-min)的值

思路:
    先将答案进行二分，然后使用dp去处理，设dp[i]表示[1,i]中能否被完整分成满足条件的几段

代码:
方法一: 通过二分的答案和k值去约束答案，如果dp[i]可行，那么标记dp[i]=1
bool check(int differ){
    memset(dp,0, sizeof(bool)*(n+2));
    int index=1;dp[0]=1;
    for(int i=1;i<=n;i++){
        while(a[i]-a[index]>differ)
            index++;
            while(i-index+1>=k){
            if(dp[index-1]){
                dp[i]=1;break;
            }
        index++;
        }
    }
    return dp[n];
}
方法二: last去标记最后一个能似的dp[i]成功的位置，如果dp[n]==n那么说明都可以分段
bool check(int differ){
    int last=0;
    for(int i=k;i<=n;i++){
        int j = dp[i-k];
        if(a[i]-a[j+1]<=differ)
            last=i;
        dp[i]=last;
    }
    return dp[n]==n;
}

2.小D的剧场(https://ac.nowcoder.com/acm/contest/369/A)
题意:
    一串音符，其中对于每三个都有限制，每个位置有49种音符可能，现在让你求总的可能方案数。

思路:
    因为每三个才有限制，第三个的方案数只取决于前两个，因此可以列出dp[500][49][49]，dp[i][j][k]表示在i种位置时
第二个位置为j，第三个位置为k的方案数，可列出: dp[t][i][j]=(dp[t][i][j]+dp[t-1][j][k])%mod;

3、最少拦截系统(hdu1257)
题意:
    有n个导弹依次发射，设置的系统第一次可以打任意高度，接下来的高度不能超过上次发射高度，求最少需要几个系统才能
将导弹全部射下。

思路:
    此题其实也不算真正意义上的dp，通过求最大LIS可获得答案，而与之对应的思想是求最大LIS的贪心+二分的方法，这种
方法不能求得最大LIS的真正意义上的值，但是可以求得最大LIS的长度。就是通过这个方法不断更新，加入求得最后答案。因
```

为每个导弹肯定是去寻找已发射导弹中离他高度最近的导弹，如果没有导弹满足条件，那么就再设一个。从而获得最后答案。

3、Max Sum Plus Plus(hdu1024)
题意:
给你a[N]数组，(1<=n<=1e6),求k个区间段最大值为多少。
思路:
设dp[i][j]表示i段区间，j结尾的最大值(j纳入其中)，因此我们可以得到状态转移方程:
dp[i][j]=max{dp[i][j-1],max{dp[i-1][t](i-1<=t<=j-1)}}+a[j].前者因为dp[i][j-1]的值已知，若不增加i，只需要
将a[j]并入最后一个区间即可。后者是还缺一个区间，将a[j]视为独立区间并入其中。但是有2个max显得处理起来极其繁琐，
且(1<=n<=1e6)，因此我们可以设w[i][j]=max{dp[i][t](i<=t<=j)}=max{w[i][j-1],dp[i][j]},从而我们可以将方程化为:
1)dp[i][j]=max{dp[i][j-1],w[i-1][j-1]}+a[j] 2)w[i][j]=max{dp[i][j],w[i][j-1]} 通过观察发现可以使用滚动数
组将其空间大小进行优化，优化结果为: 1)dp[j]=max{dp[j-1],w[t^1][j-1]}+a[j] 2)w[t^1][j]=max{dp[j],w[t^1][j-1]}
针对此数组对于时间的取值上发现，取m段需要n个数，取m-1段需要n-1个数，取m-2段需要n-2个数...因此在取m-(m-i)=i段的时候，
只需要求解n-(m-i)个数组即可

核心代码:
```
scanf("%d",&n);
for(int i=1;i<=n;i++){
    scanf("%d",&a[i]);
    pre[i]=pre[i-1]+a[i];
    w[0][i]=0;
}
int t=1;
for(int i=1;i<=k;i++){
    dp[i]=w[t][i]=pre[i];
    for(int j=i+1;j<=n+i-k;j++){
        dp[j]=max(dp[j-1],w[t^1][j-1])+a[j];
        w[t][j]=max(dp[j],w[t][j-1]);
    }
    t^=1;
}
printf("%lld\n",w[t^1][n]);
```

3、Phalanx(hdu2859)
题意:
给你一个n*n的矩阵,(1<=n<=1000),求该矩阵中最大的对称矩阵，对称先为左下角到右上角。

思路:
对对称线进行dp,设dp[i][j]表示由(i,j)点为左下角点的最大对称矩阵边长。若dp[i][j]=q,那么显然以(i,j)向上和向右q-1个对应
点都是相同的，因此dp[i][j]的值可以由dp[i-1][j+1]为保证，即(i-1,j+1)构成的为已知内部是对称矩阵，那么只需要扫一遍剩下的对应
点即可。

反思:
写题目的时候要多思考一下反向，因为出题者为了增加难度经常反向出题。

4、Making the Grade(poj3666)
题意:
给你一个a[n]数组，每次可以对一个位置上的数进行+1或者-1，求最小操作次数使得数组最后成为一个单调不减或者单调不增序列。

思路:
    通过贪心考虑，显然数组的最大值一定会出现在原a[n]数组之中，而a[n]数组最大值为1e9,我们可以通过离散化考虑这个问题，设
dp[i][j]表示到第i个数字的时候，最大值为第j小数字的最小花费费用。得dp[i][j]=min(dp[i-1][k](1<=k<=j))+abs(a[i]-b[j]).
其中b[n]为原数组a[n]重新从小到大排序后的数组，这样可以保证dp[i][j]在最大值为j大小的时候所花费的费用最小。这里使用了离散
化思想。

反思:
    对于数据较为杂乱，难以处理的时候可以将数据进行离散化，从1-n排序上去。

5、Jury Compromise(poj1015)
题意:
    有评审员会打出两个值，pi和di(0<=pi,di<=20).现在有n(1<=n<=200)个评审员,要选取m(1<=m<=20),其中要满足Pi=sigma(pi),
Di=sigma(di),要求|Pi-Di|最小，如若有相同则Pi+Di越大越好,并输出方案。

思路:
   最初的想法是设dp[i][j][2],i表示选到第i个评审员，包含i在内共有j个，0表示差和，1表示加和。因为|Pi-Di|不存在最优子结构，
比如前者有10和-10，因为-10的Pi+Di比10的大，选择了-10的，但是当下次遇到-5的时候显然应该取上10的，应该这个方法无法成立。
   换种思路，因为20*20=400，因此极限差值为[-400,400],我们可以将其提升到[0,800]进行背包处理。设state[i][j]:

```
typedef struct {
    int sum;
    bool vaild;
    int path[M];
}node;
```

state[i][j]表示有i个评审员时，差和为j的情况。其中sum记录最大值,vaild记录该方案是否存在,path[M]记录到该方法的操作顺序。
接下来的操作和背包一样，最后按照题目要求输出答案即可。

核心代码:

```
int dp() {
    for (int i = 1; i <= n; i++) {
        int sub = a[i][0] - a[i][1];
        int add = a[i][0] + a[i][1];
        for (int j = m; j >= 1; j--) {
            for (int x = 0; x <= 800; x++) {
                if (state[j - 1][x].vaild&&x + sub >= 0 && x + sub <= 800) {
                    if (state[j - 1][x].sum + add >= state[j][x + sub].sum) {
                        state[j][x + sub].sum = state[j - 1][x].sum + add;
                        state[j][x + sub].vaild = 1;
                        for (int index = 1; index < j; index++)
                            state[j][x + sub].path[index] = state[j - 1][x].path[index
    ];
                        state[j][x + sub].path[j] = i;
                    }
                }
            }
        }
    }
    for (int i = 0; i <= Base; i++) {
        if (state[m][i + Base].vaild || state[m][Base - i].vaild) {
            int sumMax = -1;
            if (state[m][i + Base].vaild)
                sumMax = state[m][i + Base].sum;
```

```
            if (state[m][Base - i].vaild&&state[m][Base - i].sum > sumMax)
                return Base - i;
            return Base + i;
        }
    }
}
```

反思:

dp要考虑是否存在最优子结构，如果没有最优子结构是行不通的.并且数据量小的时候可以考虑背包。

6、Blank(http://acm.hdu.edu.cn/showproblem.php?pid=6578)

题意:

现在有n,m(1<=n,m<=100),现在可以填4个数字0,1,2,3.现在有m个要求,即[l,r]区间内只能出现x种数字,n数列组成的方案数

思路:

设dp[2][N][N][N],表示数字[0,1,2,3]最后一次出现的位置,dp[i][j][k][q]表示排序后i>j>k>q.那么我们可以得到四种转移方程

```
dp[0][0][0][0]=1;
for(int i=1,p = 1; i <= n; i++, p ^= 1){
    for (int j = 0; j <= i; j++)
        for (int k = 0; k <= j; k++)
            for (int q = 0; q <= k; q++)
                dp[p][j][k][q] = 0;
    for(int j=0;j<i;j++) {
        for (int k = 0; k <= j; k++) {
            for (int q = 0; q <= k; q++) {
                Mod(dp[p][j][k][q]+=dp[p^1][j][k][q]);
                Mod(dp[p][i-1][k][q]+=dp[p^1][j][k][q]);
                Mod(dp[p][i-1][j][q]+=dp[p^1][j][k][q]);
                Mod(dp[p][i-1][j][k]+=dp[p^1][j][k][q]);
            }
        }
    }
    for(int j=0;j<i;j++){
        for(int k=0;k<=j;k++){
            for(int q=0;q<=k;q++){
                for(pii t:v[i]){
                    if(1+(j>=t.first)+(k>=t.first)+(q>=t.first)!=t.second)
                        dp[p][j][k][q]=0;
                }
            }
        }
    }
}
ll ans=0;
for(int i=0,p=n&1;i<n;i++){
    for(int j=0;j<=i;j++){
        for(int k=0;k<=j;k++){
            Mod(ans+=dp[p][i][j][k]);
        }
    }
}
```

7、炫酷雪花(https://ac.nowcoder.com/acm/contest/331/H)

题意:

小希把接下来连续的要做作业的时间分成n个单位，每个单位时间内小希都会受到ai的寒冷值侵袭，她可以选择在任何一些

时间站起来蹦蹦跳跳，以使得这个单位的寒冷值不侵袭她。小希最大能承受的寒冷程度是K，但是她想选择尽可能多的时间做作业，请你帮帮她!
小希受到的寒冷程度即为不蹦蹦跳跳的时间的寒冷值总和。要求输出最最多学习的时间和字典序最小的可行方案
1<=n<=5000,0<=k<=1e15,0<=a<=1e9

思路:
先进行贪心，从小到大排序得到最多的学习时间。设dp[i][j],表示i~n时间内，最多可以抖动j次的最小时间，那么转移方程为
dp[i][j]=min(dp[i+1][j]+a[i],dp[i+1][j-1])。然后从1->n遍历，设sum为到i位置时寒冷总和，那么存在如下关系:
1、sum+a[i]+dp[i+1][cnt]<=k,则sum+=a[i] 2.sum+a[i]+dp[i+1][cnt]>k,则cnt--

7.3 数位 dp

1、不要62(hdu2089)
题意:
求出区间[n,m]中数位没有62和4的个数
思路:
1、设dp[i][j]表示最高i位时,第i为j符合条件的个数。例如dp[5][0]中包含00532。数位dp的精髓就是逐位比较，最终获得答案。
设count(n)记录[0,n)的个数，因此要获得[n,m]中的个数，需要count(m+1)-count(n)。其中dp[i][j]的状态转移方程为:
1)、dp[i][j]=0 (j==4) 2)、dp[i][j]=sigma(dp[i-1][j]) (j=0,1...,9),当j=6时,需要减去dp[i-1][2]
之后就是将数位拆开，挨个去比较比如102,拆出来为201,此时先去保存100以下的数字，之后去保存102以下，100以上的数字。
最终即可获得答案。

代码:
1、

```
void init(){
     dp[0][0]=1;
     for(int i=1;i<10;i++){
    for(int j=0;j<10;j++){
        if(j==4){
            dp[i][j]=0;
        }else if(j!=6){
            for(int k=0;k<=9;k++)
                dp[i][j]+=dp[i-1][k];
        }else if(j==6){
            for(int k=0;k<=9;k++)
                dp[i][j]+=dp[i-1][k];
            dp[i][j]-=dp[i-1][2];
        }
      }
      }
}
int v[N+10];
ll solve(int di){
   ll res=0;
   v[0]=0;
   while(di){
    v[++v[0]]=di%10;
    di/=10;
   }
   v[v[0]+1]=0;
   for(int i=v[0];i>=1;i--){
    for(int j=0;j<v[i];j++){
```

```
        if(j!=4&&!(j==2&&v[i+1]==6))
            res+=dp[i][j];
    }
    if(v[i]==4)
        break;
    if(v[i]==2&&v[i+1]==6)
        break;
    }
    return res;
}

2、Beautiful numbers(https://codeforces.com/problemset/problem/55/D)
题意:
    一个正整数，如果它能被数位上每个非零数整除，那么这个数为完美数，求区间[n,m]的完美数个数。

思路:
    1、如果一个数能被它所有非0数位整除那么这个数一定被lcm{[1,9]}整除。
    2、存在定理a%(x*n)%x=a%x
    3、[1,9]所有的组合的最小公倍数，最大值为2520，共有48个，2520%lcm[任意组合]=0。
    综上关系，我们可以发现我们需要找到所有满足a%(x*n)%x=a%x=0的结果，显然本题中x*n取值为2520，因此我们可
以将数进行拆分，然后逐层搜索，dp记录数据，设dp[20][50][2525],dp[i][j][k]表示在数位i时，经过离散化过的最小
公倍数hash[lcm]=pos，k表示模后取值，因为lcm最大为2520，因此值只需取到2520即可。

核心代码:
ll dfs(int pos, int tot, int lcm, bool limit) {
   if (pos == 0) //pos表示当前位数,tot表示取模后值,lcm表示当前最小lcm，limit表示是否可以任意取值
    return (tot%lcm == 0);
   if (!limit&&dp[pos][ha[lcm]][tot] != -1)
    return dp[pos][ha[lcm]][tot];
   ll res = 0;
        int top = limit ? di[pos] : 9; //判断当前有限制
   for (int i = 0; i <= top; i++) {
    res += dfs(pos - 1, (tot * 10 + i) % mod, i ? i * lcm / gcd(i, lcm) : lcm, i == di[
    pos] && limit);
   } //进一步搜索
   if (!limit) //如果没有限制,那么这个情况的值就确定下来了
    dp[pos][ha[lcm]][tot] = res;
    return res;
}

3、Beautiful numbers(https://ac.nowcoder.com/acm/contest/163/J)
题意:
    给你一个数,若数位上所有的和能整除这个数,那么这个数可以称为美丽数,求[1,n]之中有多少个美丽数,其中1
    <=n<=1e12
思路:
    因为1e12数位之中最大数位和只有9*12=108,因此我们可以通过我们可以依次循环数位和，然后使用dfs去暴力
    搜索情况。
如果存在sigma(di)==mod,那么这个数可以成立。并设dp[12][120][120],设dp[i][j][k]表示在i位时候,数位和为j,被mod
余k的数量。

核心代码:
ll dfs(int pos, int tot, int remain, bool limit) {
   if (pos == -1) //只有当数位和与mod相等时,并且正好模掉,才会有数字满足要求
    return (tot == mod && !remain);
   if (!limit&&dp[pos][tot][remain] != -1)
    return dp[pos][tot][remain];
```

```
    ll res = 0;
    int top = limit ? di[pos] : 9;
    for (int i = 0; i <= top; i++) {
     if (i + tot > mod)
         break;
     res += dfs(pos - 1, i + tot, (10 * remain + i) % mod, limit&&i == di[pos]);
    }
    if (!limit)
     dp[pos][tot][remain] = res;
    return res;
}
```

反思：

数位dp的变种中，有一类实际是靠暴力搜索去解决问题，时间复杂度难以估计，但可以通过记忆华搜索进行剪枝，减少不必要的
访问，需要人勇敢去莽。

4、B-number(hdu3652)

题意：

给你一个n(1<=n<=1000000000)，求[1,n]内所有数中含有13，且能被13整除的数。

思路：

典型的数位dp，设dp[i][j][k]，i表示第i位，j表示数%13的结果，k表示状态，其中0表示前面不含13，1表示前一位含1，2表示前面含有13，
这样进行相应的dfs即可获得答案。

核心代码：

```
ll dfs(int pos, int tot,int state, bool limit) {
    if (pos == -1)
     return (state==2 && !tot);
    if (!limit&&dp[pos][tot][state] != -1)
     return dp[pos][tot][state];
    int top = limit ? di[pos] : 9;
    ll res = 0;
    for (int i = 0; i <= top; i++) {
     int cstate = state;
     if (state == 1 && i == 3)
         cstate = 2;
     else if (state == 1 && i != 3&&i!=1)
         cstate = 0;
     else if (state == 0 && i == 1)
         cstate = 1;
     res += dfs(pos - 1, (tot * 10 + i) % 13, cstate, limit&&di[pos] == i);
     }
     if (!limit)
     dp[pos][tot][state] = res;
    return res;
}
```

反思：

最刚开始，我只是通过flag记录是否存在13，设dp[i][j]，这种方法会使一类前面数位不存在13或存在13，但是因为之前有记录而导致return，
使得出错.若是记录dp[i][j][flag]，这样会使得也许前面出现13或没出现13，因为已有保存，而使得结果出错.若是记录dp[i][j][[1,9]]，这样
使得前面是否出现13，而已有保存，使得结果出错.若记录dp[i][j][flag][[1,9]]，这样可以获得答案，但是时间复杂度会很高，因为通过记忆
化搜索的作用就减少很多.因此可以巧妙地使用题解的方法，即dp[i][j][state]，state只记录前面是否出现13，前一位没有1的存在，前一位有1
的存在，因为本题关键就是找到13的组合，因为dfs中只要一出现13，那么接下来的搜索都以存在13为基础，而前一位出现1，则可以按照后一位存在3

或存在1，以及其他数字的情况进行搜索。最差的情况就是前面state=0.

5、F(x) (hdu4734)
题意：
存在一个函数F(x)=An*2^(n-1)+An-1*2^(n-2)+...+A1*1,其中x=(AnAn-1An-2...A1),现在求x=[0,B]中,求F(x)<=F(A)的个数

思路：
显然是一道数位dp题,这题的t(1<=t<=10000),如若使用memset(dp),那么dp空间不应该开过10000,因为时间只有500ms,显然无法
有有效的开数组方法解决问题.因此我们可以换一个思路,只需要初始化一次,之后的搜索只需要继续沿用之前的数组即可.根据题解,发现
可以开dp[12][4600],dp[i][j]表示在第i个位置时,还剩下j个数字量可以继续装,类似于背包思想.至于若从0开始计数,显然说不通,且
状态不正确。

核心代码：
```
ll dfs(int pos, int tot, bool limit) {
    if (pos == -1)
        return tot>=0;
    if (!limit&&dp[pos][tot] != -1)
        return dp[pos][tot];
    int res = 0;
    int top = limit ? di[pos] : 9;
    for (int i = 0; i <= top; i++) {
        int ntot = tot - i * (1 << pos);
        if (ntot <0)
            break;
        res += dfs(pos - 1, ntot, limit&&di[pos] == i);
    }
    if (!limit)
        dp[pos][tot] = res;
    return res;
}
```

反思：
对于因为初始化时间复杂度过高的情况,可以考虑通过只初始化一次,将接下来沿用之前的状态保存.

6、Balanced Number(hdu 3709)
题意：
一个数如果在数位中有一个位置满足诸如4139,取3为中轴,有2*4+1*1==9*1,则此数为平衡数.求[n,m]之中有多少个平衡数

思路：
遍历每个位置为中轴,设dp[20][20][2000],其中dp[i][j][k]表示数位i,中轴为j,从左开始算起到右的总平衡值,如若
tot==0,说明存在平衡.

核心代码：
```
ll dfs(int pos, int tot, int pivot, bool limit) {
   if (pos == -1)
    return tot==0;
   if (!limit&&dp[pos][pivot][tot] != -1)
    return dp[pos][pivot][tot];
   ll res = 0;
   int top = limit ? di[pos] : 9;
   for (int i = 0; i <= top; i++) {
    int tmp = tot + i * (pos - pivot);
    if (tmp < 0)
```

```
        break;
    res += dfs(pos - 1, tmp, pivot, limit&&di[pos] == i);
   }
   if (!limit)
    dp[pos][pivot][tot] = res;
   return res;
}
```

反思：
只想着如何拆分两个，其实只需要两个合并在一起即可，要心中牢记若是拆开失败，就想着如何合并

7、Balanced Number(https://vjudge.net/contest/285467#problem/K)
题意：
如果一个数它出现的位数中，奇数出现的个数为偶数，偶数出现的个数为奇数，例如6222，那么称为平衡数。求区间[n,m]中有多少个
平衡树。

思路：
因为总共有3种状态，即一个数出现0次，奇数次，偶数次.那我们可以使用三进制进行状态压缩，那么[0,9]的情况共有3**10，因此可以
开一个dp[22][60000]，dp[i][j]表示在第i个位置，j状态下有多少种可能。

核心代码：

```
ll dfs(int pos,int state,bool limit){
    if(pos==-1)
        return judge(state);
    if(!limit&&dp[pos][state]!=-1)
        return dp[pos][state];
    ll res=0;
    int top=limit?di[pos]:9;
    for(int i=0;i<=top;i++){
        res+=dfs(pos-1,(state==0&&i==0)?0:change(state,i),limit&&di[pos]==i);
    }
    if(!limit)
        dp[pos][state]=res;
    return res;
}
```

反思：
数位dp关键是状态的保存，之前一直想不到如何解决，就是因为之前走过的状态无法保存，最初有想到状态压缩保存进制，但是只考虑到了
二进制，难以保存难以保存状态，因此可以考虑3进制，这样就可以方便地保存3种状态，进行判断，今后做题如果状态偏多，可以考虑多进制
操作。

8、Seven Segment Display(zoj3962)
题意：
有一个时钟以十六进制八位显示，如FFFFFFFF，但是转化需要能量，现在给你时间，每秒十六进制加一，求最终需要消耗多少能量。
思路：
这题可以转化为数位dp，设dp[10][80]，dp[i][j]表示在i位时，前面消耗总能量为j时花费的总能量为多少。设a，b，b=a+n-1，若
b超出了8位十六进制，只需转化为solve(16^8)+solve(b)-solve(a-1)即可。
反思：
最初想到的是dp[i][j][k]，j表示数位，k表示能量，方程意义是满足满足条件的个数，但是有bug至今也不清楚，但是明显这个可以
简化，因为无论当前为是什么数，对最终结果没有任何影响，所以应该考虑清楚再写。如果一个思路不通，可以寻找是否再存其他意义的
状态方程。

8、吉哥系列故事——恨7不成妻(hdu4507)
题意:
求一个区间[N,M]内所有的满足条件的数的平方和,其中满足的条件有:1、数位中不能存在7. 2、数位整数的每一位加起来的和不是7
的整数倍. 3、这个整数不是7的整数倍.

思路:
设node dp[i][j][k]表示i位时,位数和模为j,整数模为k,node中包含cnt表示个数,sum表示每个数的和,sqrsum表示平方和.本题的
关键发现(x1+x2+...+xn)^2=x1^2+2*x1*(x2+...xn)+(x2+...+xn)^2=x1^2+2*x1*(x2+...xn)+x2^2+2*x2*(x3+...xn)+
(x3+...xn)^2=.... 无限递归下去。

核心代码:
```
node dfs(int pos, int tot, int tot2, bool limit) {
     node res; res.cnt = res.sqrsum = res.sum = 0;
     if (pos == -1) {
    res.cnt = (tot != 0 && tot2 != 0);
    return res;
     }
     if (!limit&&dp[pos][tot][tot2].cnt != -1)
    return dp[pos][tot][tot2];
     int top = limit ? di[pos] : 9;
     for (int i = 0; i <= top; i++) {
        if (i == 7)
        continue;
    node tmp = dfs(pos - 1, (tot + i) % 7, (tot2 * 10 + i) % 7, limit&&di[pos] == i);
    ll A = p[pos] * i%mod;
    res.cnt = (res.cnt + tmp.cnt) % mod;
    res.sum = (res.sum + tmp.sum + A*tmp.cnt%mod) % mod;
    res.sqrsum = (res.sqrsum + A * A%mod*tmp.cnt%mod + tmp.sqrsum + 2 * A%mod*tmp.sum%mod) % mod;
     }
     if (!limit)
    dp[pos][tot][tot2] = res;
     return res;
}
```

反思:
本题关键在于难以处理和,其实要是耐心推导公式可以发现规律,另外dp方程不一定只能有一个数值,可以定义一个结构体去
保存更多状态.

9、pair(https://ac.nowcoder.com/acm/contest/887/H)
题意:
给你A,B,C,现在求有多少个(x,y)满足x&y>c||x^y<c,其中(1<=x<=A,1<=y<=B,1<=A,B,C<=1e9)

思路:
是一个数位dp的题目,为了简化问题我们可以先取反,即答案变为A*B-cnt(x&y<=c&&x^y>=c)+max(0ll,A-C+1)+max(0ll,B-C+1),因为x,y
都大于0,因此需要减去x=0和y=0的情况.那么我们可以设dp[32][2][2][2][2],dp[i][j][k][l][m],其中i表示第i位,j表示AND限制是否存在,
k表示XOR限制是否存在,l表示x是否有限制,m表示y是否有限制.

代码:
```
ll dfs(int len,int AND,int XOR,int limit1,int limit2){
    if(len<0)
        return 1;
```

```
    if(dp[len][AND][XOR][limit1][limit2]!=-1) return dp[len][AND][XOR][limit1][limit2];
    int top=limit1?(A>>len)&1:1;
    int top2=limit2?(B>>len)&1:1;
    int c=(C>>len)&1;
    ll cnt=0;
    for(int i=0;i<=top;i++){
        for(int j=0;j<=top2;j++){
            if((!AND||(i&j)<=c)&&(!XOR||(i^j)>=c)){
                cnt+=dfs(len-1,AND&&(i&j)==c,XOR&&(i^j)==c,limit1&&i==top,limit2&&j==
    top2);
            }
        }
    }
    return dp[len][AND][XOR][limit1][limit2] = cnt;
}

int main(){
    int t;
    cin>>t;
    while(t--){
        cin>>A>>B>>C;
        memset(dp,-1, sizeof(dp));
        cout<<(A*B-dfs(30,1,1,1,1)+max(0ll,A-C+1)+max(0ll,B-C+1))<<endl;
    }
    return 0;
}
```

7.4 概率 dp

规律总结:
1、期望可以分解成多个子期望的加权和，权为子期望发生的概率，即E(aA+bB...)=aE(A)+bE(B)+...+1
2、期望从后往前找，一般dp[n]=0,dp[0]是答案
3、解决过程，找出各种情况乘上这种情况发生的概率，求和

1、Favorite Dice (spoj)
题意:
    甩一个n面的骰子，问每一面都被甩到的次数期望是多少?

思路:
    设dp[i]表示取了i种数时还需要数的期望,显然dp[n]=0,求解dp[0]为多少.本题很神奇,正着推死活推不出来,需要倒着推.
反推显然满足这个式子,dp[i]=1+i/n*dp[i]+(n-i)/n*dp[i+1],推出来为dp[i]=dp[i-1]+n/(n-i),扫一遍[n-1,0]区间即可.

核心代码:

```
dp[n] = 0;
for (int i = n - 1; i >= 0; i--)
    dp[i] = dp[i + 1] + 1.0*n /(n - i);
printf("%.2lf\n", dp[0]);
```

2、LOOPS (hdu3853)
题意:
    一个女生走迷宫，只有向下和向右走和保持在原地，每走一次会消耗掉2的魔力，求从(1,1)成功走到(r,c)所花魔力的期望。

思路:
    设dp[r][c]=0，那么有满足状态转移方程 dp[i][j] = p1*dp[i][j]+p2*dp[i][j+1]+p3*dp[i+1][j]+2

转化成 dp[i][j] = (p2*dp[i][j+1]+p3*dp[i+1][j]+2)/(1-p1) 即可

核心代码:

```
for (int i = r; i >= 1; i--) {
    for (int j = c; j >= 1; j--) {
    if (i == r && j == c)
        continue;
    if (p[i][j].p1 == 1)
        continue;
    dp[i][j] = (p[i][j].p2*dp[i][j + 1] + 2 + p[i][j].p3*dp[i + 1][j]) / (1 - p[i][j].
    p1);
    }
 }
```

反思:

dp[i][j]表示在(i,j)这个点时，还差多少魔力值才能到达终点的期望。

最开始的时候dp[i][j]将未来的状态也加入其中，其实那种想法是错误的，因为当前还未确定，未来也不会确定，因此状态转移方程所含

的应该是当前不确定的dp[i][j]和之前已经推导出的状态来推导出dp[i][j]最终推导dp[1][1]的时候就是答案。

3、King Arthur's Birthday Celebration(poj3682)

题意:

有个国王过生日，投硬币，投到k次正面朝上就结束宴会，正面朝上概率为p，每一天投一次，花钱量1、3、5、7...这样递增。求

结束宴会的期望天数和期望花钱量。

思路:

设dp[i]表示投中i次后，还需要天数的期望。显然满足 dp[i]=(1-p)*dp[i]+p*dp[i+1]+1 => dp[i]=dp[i+1]+1/p。

dp2[i]表示投中i次后，还需要花钱的期望，显然满足 dp2[i]=(1-p)*(dp2[i]+2*(dp[i]+1)-1)+p*(dp2[i+1]+2*(dp[i+1]+1)-1).

=> 得到 dp2[i]=dp2[i]+2dp[i+1]+1+(1-p)(2dp[i]+1)/p。然后dp[n]=dp2[n]=0,for循环一下就可以得到答案。

也可以正推,显然满足 dp[i]=(1-p)*dp[i]+p*dp[i-1]+1 => dp[i]=dp[i-1]+1/p,

dp2[i]=(1-p)*(dp2[i]+2*dp[i]-1)+p*(dp2[i+1]+2*dp[i+1]-1)+2.然后dp[0]=dp2[0]=0,for循环一下就可以得到答案。

反思:

概率题可以用dp方式求得,两个不同的概率题也可以用dp方程之间建立联系.

4、烟花 (https://ac.nowcoder.com/acm/contest/180/B)

题意:

小a有n个烟花，每个烟花代表着互不相同的颜色，对于第i个烟花，它有pi的概率点燃，现在小a要去点燃它们，

他想知道产生颜色的期望个数及产生恰好产生k种颜色的概率。

思路:

第一个就是简单期望相加，即E(x)=sigma(pi)

1、设方程dp[k][N],那么dp[i][j]表示有j个烟花，其中i个燃放的概率。初始化dp[0][0]=1,dp[0][i]=

(1-p[1])(1-p[2])...(1-p[i])。那么就有dp[i][j]=dp[i-1][j-1]*p[j]+dp[i][j-1]*(1-p[j]),那么

dp[k][n]即为所求答案。

2、设方程dp[N][K],那么dp[i][j]表示有i个烟花，其中有j个被燃放的概率。因为此循环使用的是for(N),

因此只用记录n-1的状态即可。因此可以使用滚动数组，设o=0，初始状态为dp[o][0]=1-p[1],dp[o][1]=p[1]。

状态转移方程为dp[o][j]=dp[o^1][j]*(1-p[i])+dp[o^1][j-1]*p[i],其中j<=min(i,k)即可。

5、流星雨(https://ac.nowcoder.com/acm/contest/368/C)

题意:

现在一共有n天，第i天如果有流星雨的话，会有wi颗流星雨。第i天有流星雨的概率是pi。如果第一天有流星雨了，

那么第二天有流星雨的可能性是p2+P，否则是p2。相应的，如果第i-1 (i≥2)天有流星雨，第i天有流星雨的可能性是
pi+P，否则是pi。求n天后，流星雨颗数的期望。

思路:
    设dp[i]表示第i天会发生流星雨的概率，那么dp[i]只和dp[i-1]有关，满足: dp[i]=dp[i-1]*(pi+p)+(1-dp[i-1])*p。然后只需乘上每天的流星雨数量即可得到答案。

反思:
    这道题我只想着如何直接推导推导期望公式，导致推出来的公式非常耗时间，公式如下:
dp[i]=dp[i-1]+(p^(i-1)*p[1]+...+p^0*p[i])*w[i]。状态转移方程是对的，但是时间会超，因为可以考虑期望的定义
即E(x)=p[x]*w[x],我们分段求出期望最后累加就是结果。

6、One Person Game(zoj3329)
题意:
    有三个骰子，分别有k1,k2,k3个面。每次掷骰子，如果三个面分别为a,b,c则分数置0,否则加上三个骰子的分数之和。
当分数大于n时结束。求游戏的期望步数。初始分数为0

思路:
    设dp[i]表示达到i分时到达目标状态的期望，pk为投掷k分的概率，p0为回到0的概率则dp[i]=∑(pk*dp[i+k])+dp[0]*p0+1;
都和dp[0]有关系，而且dp[0]就是我们所求，为常数设dp[i]=A[i]*dp[0]+B[i];代入上述方程右边得到:
dp[i]=∑(pk*A[i+k]*dp[0]+pk*B[i+k])+dp[0]*p0+1=(∑(pk*A[i+k])+p0)dp[0]+∑(pk*B[i+k])+1;明显A[i]=(∑(pk*A[i+k])+p0)
B[i]=∑(pk*B[i+k])+1,先递推求得A[0]和B[0]. 那么dp[0]=B[0]/(1-A[0]);

7、Dice(hdu 4652)
题意:
    现在有一个m面骰子,每个面都有一个特定的数,现在有2个操作,1: 询问投出最后连续n个相同数字的期望 2: 询问投出最后n个不连续
数字的期望.

思路:
    对于1情况,我们可以设dp[n]=0,有关系式dp[i]=1+dp[i+1]/m+(m-1)*dp[1]/m,通过dp[i+2]-dp[i+1]=m*(dp[i+1]-dp[i]),可以得到
dp[0]-dp[1]=1,dp[1]-dp[2]=m,...,dp[n-1]-dp[n]=m^(n-1),可以得到dp[0]=(m^n-1)/(m-1)
    对于2情况,要注意是连续n个数不相同,因此dp[n]=0,dp[i]=1+(m-i)*dp[i]/m+(dp[1]+dp[2]+...dp[i])/m,因为如果加入相同的数,那么
会随着数字的不同回到的原始点不同,因此有了(dp[1]+dp[2]+...dp[i])/m,方法如上递推即可.

8、Maze(hdu 4035)
题意:
    有n个房间,由n-1条隧道连通起来,实际上就形成了一棵树,从结点1出发,开始走,在每个结点i都有3种可能:1.被杀死,回到结点1处（概率为ki）
2.找到出口，走出迷宫 （概率为ei） 3.和该点相连有m条边，随机走一条. 求: 走出迷宫所要走的边数的期望值。

思路:
    设dp[i]=Ai*dp[1]+Bi*dp[fa[i]]+Ci,re[i]=1-ki-ei,other=re[i]/m
    关于叶子节点: dp[i]=ki*dp[1]+Ei*0+other*(dp[fa[i]]+1)=ki*dp[1]+other*dp[fa[i]]+other,因此可以得到叶子节点:
Ai=ki,Bi=other,Ci=other
    关于非叶子节点:dp[i]=ki*dp[1]+Ei*0+other*(dp[fa[i]]+1+sigma(dp[child[i]]+1)),设j=nx[i],则sigma(dp[child[i]]+1)
=sigma(Aj)*dp[1]+sigma(Bj)*dp[fa[j]]+sigma(Cj),带入原式可以得到Ai=(ki+other*sigma(Aj))/(1-other*sigma(Bj)),
Bi=other/(1-other*sigma(Bj)),Ci=(re[i]+other)/(1-other*sigma(Bj))
    因此一个dfs即可.

代码:

```
bool dfs(int from,int pre){
    int m = g[from].size();
    double other = re[from]/m;
    double a = k[from],b = 1,c = re[from];
    for(int i=0;i<m;i++){
        int to = g[from][i];
        if(to==pre) continue;
        if(!dfs(to,from)) return false;
        a+=other*A[to]; b-=other*B[to]; c+=other*C[to];
    }
    if(b<eps) return false;
    A[from]=a/b; B[from]=other/b; C[from]=c/b;
    return true;
}
```

9.Activation

题意:

Tomato在排队激活游戏,有四种情况:1.注册失败,但是不影响队列顺序,概率为p1 2.连接失败,队首的人排到队尾,概率为p2 3.注册成功,队首离开队列,概率为p3 4.服务器崩溃,激活停止,概率为p4,现在给出总排队人数n,Tomato排在第m个,一个数k，然后是四种情况的概率p1-p4;如果Tomato前面在k-1个人之内,并且服务器崩溃了,那么这种情况Tomato认为服务器是很low的.问你,发生这种很low的情况的概率。

思路:

设dp[i][j]为目前有i个人排队,Tomato排在第j个位置发生这种情况的概率.

当j=1时，dp[i][j] = p1*dp[i][j]+p2*dp[i][i]+p4;

当1<j<=k时，dp[i][j]= p1*dp[i][j]+p2*dp[i][j-1]+p3*dp[i-1][j-1]+p4;

当k<j<=i时，dp[i][j] = p1*dp[i][j]+p2*dp[i][j-1]+p3*dp[i-1][j-1];

化简得:

当j=1时，dp[i][j] = p21*dp[i][i]+p41;

当1<j<=k时，dp[i][j]= p21*dp[i][j-1]+p31*dp[i-1][j-1]+p41;

当k<j<=i时，dp[i][j] = p21*dp[i][j-1]+p31*dp[i-1][j-1];

其中: p21=p2/(1-p1)、p31=p3/(1-p1)、p41=p4/(1-p1);

循环i: 1-n;

由上面的式子可以看出，求dp[i][j]的时候dp[i-1][j-1]是已经计算出来了的。我们不妨把后面的部分用c数组保存起来，得到:

当j=1时，dp[i][j] = p21*dp[i][i]+c[1];

当1<j<=k时，dp[i][j]= p21*dp[i][j-1]+c[j]，其中，c[j]=p31*dp[i-1][j-1]+p41;

当k<j<=i时，dp[i][j] = p21*dp[i][j-1]+c[j]，其中c[j]=p31*dp[i-1][j-1];

显然，dp[i][1]与dp[i][i]有关，而dp[i][j]又与dp[i][j-1]有关，这样就形成了一个环。所以，我们先利用上面3个式子迭代求出dp[i][i]:

dp[i][i]=dp[i][i]*p21^i+c[1]*p21^i-1+c[2]*p21^i-2+......+c[i]; 变个形即可求出dp[i][i]

得出dp[i][i]，那么dp[i][1]也可以得出，之后就递推就行了。

//

3、Kids and Prizes(sgu495)

题意:

本题就是有n个奖品，m个人排队来选礼物，对于每个人，他打开的盒子，可能有礼物，也有可能已经被之前的人取走了,

然后把盒子放回原处。为最后m个人取走礼物的期望。

思路:

1、设每种礼物不会被拿到的概率为((n-1)/n)^m,因此不会被拿到的期望为n*((n-1)/n)^m，所以会拿到的期望为n(1-((n-1)/n)^m)

2、设dp[i]为总共i个人取得礼物时所拿到奖品的期望，则dp[1]=1,显然满足状态转移方程:

dp[i] = dp[i-1] + (n-dp[i-1])/n*1+dp[i-1]/n*0 => dp[i] = dp[i-1] + (n-dp[i-1])/n

反思:
概率题推期望不是只有一种模式逆推，要结合具体情况，通过赋予对状态转移方程的实际意义来推导。对于概率的题如果实在推不出来
可以尝试使用日常经验的手段去推。

5、Where is the canteen(hdu2262)
题意:
在一个迷宫中，寻找餐厅，餐厅有多个，每次上下左右走向一个空地，概率为等可能，求到达餐厅的步数期望

思路:
显然此题满足概率dp,状态转移方程满足 dp[i][j]=p1*dp[i+1][j]+p2*dp[i-1][j]+p3*dp[i][j-1]+p4*
dp[i][j+1]+1,其中点为餐厅的dp[i][j]=0，求最后dp[startx][starty]为多少？因此只需要先bfs，然后使用高斯消元即可获得答案。 关键在于转移方程的理解。dp[i][j]表示到达餐厅还剩下多少步，因此它的步数是由其余四个防线的dp[i][j]*p+移动步数1即可。

反思：代码不够优美，要多看看优美的代码

7.5 斜率 dp

1、Print Article(hdu 3507)
题意:
给出N个单词，每个单词有个非负权值Ci，现要将它们分成连续的若干段，每段的代价为此段单词的权值和，还要加一个常数M,
即(∑Ci)^2+M。现在想求出一种最优方案，使得总费用之和最小。

思路:
斜率dp裸题,n=500000,设dp[i]表示[1,n]的最优结果,显然满足dp[i]=min{dp[k]+sigma(c[k+1],c[i])+M},显然斜率优化即可.

2、Lawrence(hdu2829)
题意:
给出一条笔直无分叉的铁路上有n个仓库，每个仓库有一个v[i]代表价值,每两个仓库之间算作一段铁路，现在有m次攻击机会，一次攻
击可以炸毁一段铁路;m次攻击后，剩余的总价值为: Σ(v[i]*v[j]),i和j为所有任意两个互相可到达的仓库.现要求选定m段铁路进行攻击炸毁,
然后使得总价值最小.

思路:
设dp[i][j]是前i个仓库,炸掉j段铁路后,剩余总价值的最小值.(显然,j<i)设w[a][b]表示铁路完好的情况下，从a仓库到b仓库的总价值,
那么,就有:dp[i][j]=min(dp[k][j-1]+w[k+1][i]),j≤k<i; 方程的意义是: 炸毁仓库k和仓库k+1之间的那段铁路（即第k段铁路），算出总
价值，枚举k找到最小的.
那么如何计算w[k+1][i]呢? w[1][i]=w[1][k]+w[k+1][i]+(v[1]+v[2]+…+v[k])×(v[k+1]+v[k+2]+…+v[i])=w[1][k]+w[k+1][i]+
sum[k]×(sum[i]-sum[k]),即w[k+1][i]=w[1][i]-w[1][k]-sum[k]×(sum[i]-sum[k])
我们把w[k+1][i]的计算式带入状态转移方程得到:dp[i][j]=min{dp[k][j-1]+w[1][i]-w[1][k]-sum[k]×(sum[i]-sum[k])}
那么，对于这个DP，j一个循环、i一个循环、k一个循环，就是O(n3)的时间复杂度;需要斜率优化，优化到O(n2)即可。

3、Cross the Wall(Uvalive 5097)
题意:

有N个长方形，要穿过一张纸，最多可以在这张纸上剪掉K个长方形，剪掉一个长方形的代价为该长方形的长*宽，现在问所有长方形都通过的
最小代价

思路:
首先,排除掉那个无用的长方形,无用的长方形指的是h[i]<h[j]且w[i]<[j](h指长,w指宽)接着排序,排序按照长递减,宽递增,那么拥有单调性
后可以列方程了,对于方程dp[i][j]表示最多有i个洞,j个矩形的最优值,因此即使i>1,也可以从1开始遍历.

代码:
```
sort(p+1,p+1+n,cmp);
int tot=1;
for(int i=2;i<=n;i++){
    if(p[i].w<=p[tot].w) continue;
    p[++tot]=p[i];
}
n=tot;
for(int i=1;i<=n;i++) dp[1][i]=p[1].h*p[i].w;
ll ans=dp[1][n];
m=min(m,n);
for(int k=2;k<=m;k++){
    int tail=0,head=0;
    q[tail++]=0; dp[k][k]=0;
    for(int i=1;i<=n;i++){
        while (tail>1+head&&isOK(k,i,q[head+1],q[head])) head++;
        dp[k][i]=getsum(k,i,q[head]);
        while (tail>1+head&&isOK2(k,i,q[tail-1],q[tail-2])) tail--;
        q[tail++]=i;
    }
    ans=min(ans,dp[k][n]);
}
printf("%lld\n",ans);
```

4、Picnic Cows(hdu3045)
题意:
给你一些牛，把它们分成若干组，每一头牛有自己的价值，每一组的牛的个数不少于T，每一组贡献的价值为这一组内的牛与最小
价值牛的差的和，问所有组贡献的价值最小是多少。

思路:
这题dp方程容易想出dp[i]=dp[k]+sum[i]-sum[k]-(i-k)*a[k+1];(k>=m && i-(k+1)+1>=m)，但是在处理的时候有一定的技巧
```
int j = i-t+1; if(j<t) continue; while(tail>1+head&&isOK2(j,q[tail-1],q[tail-2])) tail--; q[tail++]=j;
```

5、Tree Construction(hdu3516)
题意:
平面上有点,每次只能向上和向右连接,每次连接的长度就是代价,求把n个点连接成一棵树的最小代价

思路:
f[i][j]表示把[i,j]中的点合成一棵树的最小代价,f[i][j]=min{f[i][k]+f[k+1][j]+abs(x[k+1]-x[i])+abs(y[k]-y[j])}
然后利用四边形不等式进行优化,需要注意的是,这个dp应该先枚举区间长度，初始时s[i][i]=i

6、Post Office(1160)
题意:
一条高速公路，有N个村庄，每个村庄均有一个唯一的坐标，选择P个村庄建邮局，问怎么选择，才能使每个村庄到其最近邮局的距离
和最小？最后打印这个最小值。

思路:
dp[i][j]表示有i个邮局,前j个村庄的最优结果,对于dp[i][j]=min{dp[i-1][k]+w[k+1][j]}的情况,对于这种式子可以使用四边形不等式优化,对于这种情况优化的不等式为in[i-1][j]<=in[i][j]<=in[i][j+1],并且需要逆推,因为in[i][j+1]的存在,所以第二维度从for(n->i),并且需要注意越界的情况in[i][v+1]=v-1;

代码:

```
int main() {
    while(scanf("%d%d",&v,&p)==2){
        for(int i=1;i<=v;i++) {
            scanf("%lld",&a[i]);
            s[i]+=s[i-1]+a[i];w[i][i]=0;in[i][i]=i;
        }
        for(int len=2;len<=v;len++){
            for(int i=1;i+len-1<=v;i++){
                int j=i+len-1;
                w[i][j]=INF; int index;
                for(int k=in[i][j-1];k<=in[i+1][j];k++){
                    ll x = a[k]*(2*k-i-j+1)+s[j]+s[i-1]-2*s[k];
                    if(w[i][j]>x){
                        w[i][j]=x; index=k;
                    }
                }
                in[i][j]=index;
            }
        }
        for(int i=1;i<=v;i++) dp[1][i]=w[1][i],in[1][i]=1;
        for(int i=2;i<=p;i++){
            in[i][v+1]=v-1; dp[i][i]=0;
            for(int j=v;j>=i+1;j--){
                dp[i][j]=INF;
                for(int k=in[i-1][j];k<=in[i][j+1];k++){
                    ll x = dp[i-1][k]+w[k+1][j];
                    if(dp[i][j]>=x) dp[i][j]=x,in[i][j]=k;
                }
            }
        }
        printf("%lld\n",dp[p][v]);
    }
}
```

7、Batch Scheduling(poj1180)
题意:
N个任务排成一个序列在一台机器上等待完成(顺序不得改变),这N个任务被分成若干批,每批包含相邻的若干任务.从时刻0开始,这些任务被分批加工,第i个任务单独完成所需的时间是Ti.在每批任务开始前,机器需要启动时间S,而完成这批任务所需的时间是各个任务需要时间的总和(同一批任务将在同一时刻完成)。每个任务的费用是它的完成时刻乘以一个费用系数Fi。请确定一个分组方案,使得总费用最小.(1<=N<=10000)

思路:
S表示启动时间,T[i]是前i个任务的时间和,C[i]是前i个任务的开销和f[i][j]=Min{f[i-1][k]+(S*i+T[j])*(c[j]-c[k])};
看了别人的结题报告,找到了优化到O(n*n)的方法。就是从n往前推。
sumT[i]表示从i到n的任务所需要的时间总和,sumF[i]表示从i到n的费用系数总和,dp[i]表示对于从i到n的任务安排的最优解。

那么很容易可以得出这样一个简单的DP状态转移方程：(注:数组存储从1到n)
    dp[i]=min{dp[j]+(S+sumT[i]-sumT[j])*sumF[i]{i<j<=n+1} 边界条件 dp[n+1] = 0
    从后往前推效率可以降一维的原因:
    正向思考,在前面的分块情况不清楚的时候是没法决定下一块的开销的,但是反过来,假设前面都没有分块,先算后面的开销,然后.
如果前面要分块，后面的开销就会全部多出来的一个S，将这个S算进当前的分块开销里面，于是倒过来动态成为可能。

8、Best Cow Fences(poj2018)
题意:
    给定一个非负序列，求长度大于F的连续子序列的平均数最大.

思路:
    在实数上二分平均数mid，判断a中是否有长度大于F平均数大于等于mid，再进行调整二分区间设定一个b数组，b[i]=a[i]-mid
当b[i]的区间和大于等于0的时候说明区间平均数大于等于mid,用sum数组表示b数组前缀和,再求出长度大于等于F的所有区间中的最大
区间和=前缀和-前面的最小前缀和(要保证区间长度大于F),判读和是否大于等于0
    第二个方式可以通过斜率优化找最优值,可以画图分析可得

代码:
```
bool isOK(double x){
    for(int i=1;i<=n;i++) s[i]=s[i-1]+cow[i]-x;
    double minn = inf;
    for(int i=f;i<=n;i++){
        minn=min(minn,s[i-f]);
        if(s[i]-minn>=0) return true;
    }
    return false;
}
```

7.6 树形 dp

1、Computer(hdu 2196)
题意:
    给出一棵树，求离每个节点最远的点的距离.

思路:
    方法一:那么我们来设列dp方程吧，我们思考当前点x的最远点距离是怎么得到的，只有两种情况：1、来自他的子树(红色部分) 2、来自他
的子树以外的树（蓝色部分简称父亲部）。第一种情况的话可以直接自底向上树形dp得到每一个节点的子树的最远点距离。 那么第二种情况就有点难办，
父亲部的最远点距离可以从哪里来呢？有两种情况：1、父亲点fa的父亲部 2、父亲点的子树
    对于第二种情况的话会有一种情况需要考虑，想到这又不大家应该也会发现，父亲部的子树可能包括红色的部分，如果我们冒冒然去继承，
那么就会造成没法继承到蓝色部分的解。怎么办呢，我们需要判断一下，假如fa的最远点路径经过了x，那么我们就不继承他，改为继承fa子树的次远点
距离。
    那么我们可以设列dp方程了，我们设f[i][0]为i节点子树的最远点距离，f[i][1]为i节点子树的次远点距离，设f[i][2]为i节点的父亲部的最远
点距离。那么我们列出dp方程:
    当x不在fa的最远点路径上: f[x][2]=max(f[fa][0],f[fa][2])+dist(x,fa)
    当x在fa的最远点路径上: f[x][2]=max(f[fa][1],f[fa][2])+dist(x,fa)

    方法二: 先从1为根节点，求得一个树直径上一点p,然后以p为根求一个树直径,确定第二个直接端点p2,那么任意一个点的最远点是max(dis[i],
 dis2[i])即可

代码:

```
int n,dp[N][3],p[N][2];
vector<pii>e[N];
void dfs(int from,int pre){
    dp[from][0]=dp[from][1]=0;
    for(int i=0;i<e[from].size();i++){
        int to = e[from][i].first,w=e[from][i].second;
        if(pre==to) continue;
        dfs(to,from);
        if(dp[from][0]<dp[to][0]+w){
            dp[from][1]=dp[from][0];
            dp[from][0]=dp[to][0]+w;
            p[from][0]=to;
        }else if(dp[from][1]<dp[to][0]+w){
            dp[from][1]=dp[to][0]+w;
            p[from][1]=to;
        }
    }
}

void dfs2(int from,int pre){
    for(int i=0;i<e[from].size();i++){
        int to = e[from][i].first,w=e[from][i].second;
        if(pre==to) continue;
        if(to!=p[from][0]) dp[to][2]=w+max(dp[from][2],dp[from][0]);
        else dp[to][2]=w+max(dp[from][2],dp[from][1]);
        dfs2(to,from);
    }
}
```

2、Rebuilding Roads(poj1947)

题意:

给出一棵树，问现在要得到一颗有p个节点的子树，需要最少减掉几条边?

思路:

设dp[i][j]表示i为根节点的子树含j个节点最少减少几条边.进行树形dp+背包即可

代码:

```
void dfs(int from,int pre){
    memset(dp[from],0x3f, sizeof(dp[from])); dp[from][1]=0; node[from]=1;
    for(int i=0;i<e[from].size();i++){
        int to=e[from][i];
        if(pre==to) continue;
        dfs(to,from);
        node[from]+=node[to];dp[from][1]++;
    }
    for(int i=0;i<e[from].size();i++){
        int to=e[from][i];
        if(pre==to) continue;
        for(int j=m;j>=1;j--){
            for(int k=1;k<=node[to]&&j-k>=1;k++){
                dp[from][j]=min(dp[from][j],dp[from][j-k]+dp[to][k]-1);
            }
        }
    }
}
```

3、Starship Troopers(hdu1011)

题意:

给出每个房间拥有的BUG数和能得到的能量数，然后给出每个房间的联通图，要到下一个房间必须攻破上一个房间，

每个士兵最多消灭20个BUG，就算不足20个BUG也要安排一个士兵.

思路:
dp[from][j]=max(dp[from][j],dp[from][j-k]+dp[to][k]),关键在于如何确定一个跟节点一定在背包中的情况.

代码:
```
void dfs(int from,int pre){
    int r = (p[from].w+19)/20;
    for(int i=m;i>=r;i--) dp[from][i]=p[from].v;
    for(int i=0;i<e[from].size();i++){
        int to = e[from][i];
        if(to==pre) continue;
        dfs(to,from);
        for(int j=m;j>=r;j--){
            for(int k=1;j-k>=r;k++){
                dp[from][j]=max(dp[from][j],dp[from][j-k]+dp[to][k]);
            }
        }
    }
}
```

4、Find Metal Mineral(hdu 4003)
题意:
给你一颗有n个节点的树，给出每两个相连节点边的权值（如果你的一个机器人要走这条边花费的能量），再给你k个机器人，问
从s点出发，最少花费多少d能量可以遍历所有的节点。

思路:
dp[i][j]表示对于以i结点为根结点的子树，放j个机器人所需要的权值和。当j=0时表示放了一个机器人下去，遍历完结点后又
回到i结点了。状态转移方程类似背包,如果最终的状态中以i为根结点的树中有j(j>0)个机器人，那么不可能有别的机器人r到了这棵
树后又跑到别的树中去因为那样的话，一定会比j中的某一个到达i后跑与r相同的路径再回到i，再接着跑它的路径要差(多了一条i回
去的边)这样的话，如果最后以i为根结点的树中没有机器人，那么只可能是派一个机器人下去遍历完后再回来。
状态转移，使用的“分组背包”思想。使用一维数组的“分组背包”伪代码如下:
```
for 所有的组i
    for v=V..0
        for 所有的k属于组i
            f[v]=max{f[v],f[v-c[k]]+w[k]}
```
对于每个根节点root，有个容量为K的背包,如果它有i个儿子，那么就有i组物品，价值分别为dp[son][0],dp[son][1].....
dp[son][k] ，这些物品的重量分别为0,1,.....k.现在要求从每组里选一个物品（且必须选一个物品）装进root的背包，使得容量
不超过k的情况下价值最大。那么这就是个分组背包的问题了。但是这里有一个问题，就是每组必须选一个物品。对于这个的处理，我
们先将dp[son][0]放进背包，如果该组里有更好的选择，那么就会换掉这个物品，否则的话这个物品就是最好的选择。这样保证每组
必定选了一个。

代码:
```
void dfs(int from,int pre){
    memset(dp[from],0, sizeof(dp[from]));
    for(int i=0;i<e[from].size();i++){
        int to=e[from][i].first,w=e[from][i].second;
        if(pre==to) continue;
        dfs(to,from);
        for(int j=m;j>=0;j--){
            dp[from][j]+=dp[to][0]+2*w;
```

```
            for(int k=1;k<=j;k++)
                dp[from][j]=min(dp[from][j],dp[from][j-k]+dp[to][k]+k*w);
        }
    }
}
```

5、The Ghost Blows Light(hdu4276)

题意:

一个有N个节点的树形的地图，知道了每条变经过所需要的时间，现在给出时间T，问能不能在T时间内从1号节点到N节点。

每个节点都有相对应的价值,而且每个价值只能被取一次,问如果可以从1号节点走到n号节点的话，最多可以取到的最大价值为多少。

思路:

先求出从1号节点到n号节点的最短路，如果花费大于时间T，则直接输出不符合，将最短路上的权值全部赋值为0，在总时间T上

减去最短路的长度，表示最短路已经走过，对其它点进行树形背包求解，需要注意的是如果不是最短路上的边都要走两次，即走过去

还要再走回来,状态转移方程:

dp[i][j]=max(dp[i][j],dp[i][k]+dp[i][j-2*val-k])

6、Fire(poj2152)

题意:

Z国有N个城市，编号为从1到N。城市之间用高速公路连接，并且每两个城市之间都有唯一一条路径。最近Z国经常发生火灾，所以

政府决定在一些城市修建消防站。在城市K建立消防站要花费W（K）。不同的城市花费不同。如果在城市K没有消防站，则离他最近的

消防站与他的距离不能超过D（K），不同城市的D也不相同。为了省钱，政府希望你能算出修建消防站最小的总花费。

思路:

dp[i][j]表示以i为根的子树里每个节点都被消防站管理，并且城市i被城市j所建的消防站管理情况下的最小花费。best[i]表示以

i为根的子树的所有节点都被管理时的最小花费，我们的目的就是求出best[1]。而best[i]显然就是dp[i][j]中的最小值（j表示i的所有孩子）

dis[i]表示以key为中心，城市i到城市key的距离，当距离大于D（key）时，就意味着key不能由i来管理。

首先dfs到子节点，然后求出以key为中心的所有距离dis，再枚举每个节点i，考虑dp[key][i]，如果dist[i]>d[key]（即key能容忍消防

站到他的最远距离），就将dp[key][i]置为一很大的数（M=1<<30），表示该情况不会被取到。如果能取到，则在此条件下枚举key的孩子，状

态转移方程: dp[key][i]=w[i]+sum(min(best[j],dp[j][i]-w[i]))。即: 城市key被城市i管理时，其花费为w[i]与min（各孩子节点的最小花

费，孩子节点j被i管理的最小花费减去i的建设费用）。最后best[key]为dp[key][i]最小的一个。

代码:

```
int n,w[N],d[N],head[N],tot,best[N],dis[N];
int dp[N][N];
struct node{
    int to,nx,w;
}edge[N<<1];

void add_edge(int from,int to,int val){
    edge[tot].to=to; edge[tot].nx=head[from]; edge[tot].w=val;
    head[from]=tot++;
}

void DFS(int from){
    for(int i=head[from];i;i=edge[i].nx){
        int to=edge[i].to;
        if(dis[to]!=-1) continue;
```

```
        dis[to]=dis[from]+edge[i].w; DFS(to);
    }
}

void dfs(int from,int pre){
    for(int i=head[from];i;i=edge[i].nx){
        int to=edge[i].to;
        if(to==pre) continue;
        dfs(to,from);
    }
    memset(dis,-1, sizeof(int)*(n+1));
    dis[from]=0;DFS(from);
    best[from]=inf;
    for(int i=1;i<=n;i++) dp[from][i]=inf;
    for(int i=1;i<=n;i++){
        if(dis[i]<=d[from]){
            dp[from][i]=w[i];
            for(int j=head[from];j;j=edge[j].nx){
                int to=edge[j].to;
                if(to==pre) continue;
                dp[from][i]+=min(best[to],dp[to][i]-w[i]);
            }
            best[from]=min(best[from],dp[from][i]);
        }
    }
}

int main() {
    int t,x,y,z;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        for(int i=1;i<=n;i++) scanf("%d",&w[i]);
        for(int i=1;i<=n;i++) scanf("%d",&d[i]);
        memset(head,0, sizeof(int)*(n+1)); tot=1;
        for(int i=1;i<n;i++){
            scanf("%d%d%d",&x,&y,&z);
            add_edge(x,y,z); add_edge(y,x,z);
        }
        dfs(1,0);
        printf("%d\n",best[1]);
    }
    return 0;
}
```

7、GeoDefense(hdu4044)

题意:

地图是一个n个编号为1～n的节点的树，节点1是敌人的基地，其他叶子节点都是你的基地。敌人的基地会源源不断地出来怪兽，

为了防止敌人攻进你的基地，你可以选择造塔。每个节点最多只能造一个塔，且节点i可以有ki种塔供你选择，价钱和攻击力分别为

price_i，power_i，攻击力power_i，效果是让敌人经过这个节点时让敌人的血减少power_i点。那么从敌人的基地到你的任意一个叶

子基地的路径，这条路径上的所有塔的攻击力之和，就是这个基地的抵抗力。敌人的攻击路径是不确定的，为了保护你的所有基地，

你要确定所有基地中抵抗力最低的一个。 你只有数量为m的钱，问最佳方案，可以抵挡敌人的最大血量是多少？也就是，让所有叶子

基地中抵抗力最低的一个的值尽量大，最大是多少？

思路:
树形dp，dp[u][j]表示到达u点还有j块钱的最大攻击力，那么将j分配给孩子，取孩子的最小值，取分配方案的最大值就行了，
因为每个点都可以建塔，所以更新树形更新完dp[u][j]后再01背包放哪个塔来更新，注意price可以为0

代码:
```
int n,m;
vector<int>e[N];
int dp[N][M],k[N];
struct node{
    int price,val;
}p[N][505];

void dfs(int from,int pre){
    dp[from][0]=inf;
    for(int i=0;i<e[from].size();i++){
        int to=e[from][i];
        if(to==pre) continue;
        dfs(to,from);
        for(int j=m;j>=0;j--){
            int x=0;
            for(int k=0;k<=j;k++) x=max(x,min(dp[to][k],dp[from][j-k]));
            dp[from][j]=x;
        }
    }
    if(dp[from][0]==inf) dp[from][0]=0;
    for(int i=m;i>=0;i--){
        int x=dp[from][i];
        for(int j=1;j<=k[from];j++){
            if(p[from][j].price<=i){
                x=max(x,dp[from][i-p[from][j].price]+p[from][j].val);
            }
        }
        dp[from][i]=x;
    }
}

int main() {
    int t,u,v;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        for(int i=1;i<=n;i++) e[i].clear();
        for(int i=1;i<n;i++){
            scanf("%d%d",&u,&v);
            e[u].push_back(v); e[v].push_back(u);
        }
        scanf("%d",&m);
        for(int i=1;i<=n;i++){
            scanf("%d",&k[i]);
            for(int j=1;j<=k[i];j++){
                   scanf("%d%d",&p[i][j].price,&p[i][j].val);
            }
        }
        for(int i=1;i<=n;i++) for(int j=1;j<=m;j++) dp[i][j]=0;
        dfs(1,0);
        printf("%d\n",dp[1][m]);
    }
    return 0;
```

```
}
```

7.7 KMP、E-KMP、Manacher

KMP内容:

1、KMP最小循环节、循环周期:

定理: 假设S的长度为len，则S存在最小循环节，循环节的长度L为len-next[len]，子串为S[0…len-next[len]-1]。

(1) 如果len可以被len - next[len]整除，则表明字符串S可以完全由循环节循环组成，循环周期T=len/L。

(2) 如果不能，说明还需要再添加几个字母才能补全。需要补的个数是循环个数L-len%L=L-(len-L)%L=L-next[len]%L，L=len-next[len]。

注意: 对于补全最小循环节，n % (n - Next[n]) == 0 && n / (n - Next[n]) > 1 表示无需再补循环节，n - Next[n] - n % (n - Next[n])表示
补全最小循环节的最少个数。

1) 求最小循环节周期>1的情况

```
for(int i=1;i<=n;i++){
    if(i%(i-nx[i])==0&&i/(i-nx[i])>1)
        printf("%d %d\n",i,i/(i-nx[i]));
}
```

2) 求最小循环节周期

```
printf("%d\n",(n%(n-nx[n])==0)?(n/(n-nx[n])):1);
```

3) 补全最小循环节,周期要K>=2

```
if (n % (n - Next[n]) == 0 && n / (n - Next[n]) > 1){
    printf("0\n");
    continue;
}
int len = n - Next[n]; // 最小循环节
printf("%d\n", len - n % len);
```

2、KMP将一个字符串补成代价最小的回文串
    只需将字符串倒过来，进行KMP匹配，结果2*n-i即可。

```
int KMP() {
    int i, j;
    getNext();
    i = j = 0;
    while (j < n&&i<n) {
        while (i != -1 && x[i] != y[j])
            i = nx[i];
        i++;j++;
    }
    return 2*n-i;
}
```

3、求一个字符串的所有前缀在字符串中出现过多少次
    设dp[i]表示以i结尾的所有成立的字符串个数，因此状态转移返程为dp[i]=(dp[nx[i]]+1)%mod

```
for(int i=1;i<=n;i++){
    dp[i]=(dp[nx[i]]+1)%mod;
    res=(res+dp[i])%mod;
}
```

4、构造一个k([1,1e9])个字符串，字符限定来自提供的y,但是不能出现x字符串,求可以组合出来的字符串的最大数量。
    考虑到k最大为1e9,需要通过矩阵快速幂解决问题,因此关键就是构造矩阵.矩阵构造需要通过next数组,可以将矩阵长度限定为[0,n-1],其意义是
前i个字符已经匹配成功，需要匹配下一个字符。如果匹配成功则转移到下一个位置，如果匹配失败则通过next数组所转移到位置即可。

```
for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
        int t = i;
        while(t>0&&y[j]!=x[t])
              t=nx[t];
              if(y[j]==x[t]||t==-1)
                 t++;
              A.arr[i][t]++;
     }
 }
```

5、给定一个字符矩阵(1<=r<=10000,1<=c<=75),求最小的子矩阵可以将矩阵全部覆盖，运行多余。
    本题最初使用的方法是求所有的行的最小循环节和列的最小循环节,然后求最小公倍数相乘即是答案。但实际上有一组样例会否决情况，

```
Input
2 8
ABCDEFAB
ABCDEABC
2 8
ABCDEFAB
AAAABAAA
Output
16
12
```

    显然上述方法存在问题，因此得寻找其他方法。注意到1<=c<=75，我们可以设立一个vis[N]数组,vis[i]表示长度为i的循环串出现的个数，
如果出现vis[i]==r,则说明这个循环串是可行的。这样最小满足的即是最小矩阵的列大小，接下来求行大小，因为列大小确定，我们只需要对r行
字符串进行求next数组,求出他们的最小循环节即是最小行大小。精彩之处在于使用整个字符串数组进行匹配。

```
void solve_col(){
    int i,j;
    j=nx[0]=-1;
    i=0;
    while(i<r){
        while(-1!=j&&strcmp(cow[i],cow[j])!=0)
            j=nx[j];
        nx[++i]=++j;
    }
    ans_row=r-nx[r];
}
```

E-KMP内容:
    定义母串S和子串T,S的长度为n，T的长度为m;求字符串T与字符串S的每一个后缀的最长公共前缀。也就是说,设有extend数组:extend[i]
表示T与S[i,n-1]的最长公共前缀，要求出所有extend[i](0<=i<n)。
（注意到，如果存在若干个extend[i]=m，则表示T在S中完全出现，且是在位置i出现，这就是标准的KMP问题，所以一般将它称为扩展KMP算法。）

1、求一个字符串所有前缀在字符串中出现次数和
    对该字符串进行e-kmp,得到nx[n]数组，显然sum(nx[0]...nx[n-1])就是答案。

7.8 后缀数组、后缀树、后缀自动机

```
后缀数组:
    后缀就是从字符串的某个位置i到字符串末尾的子串，我们定义以s的第i个字符为第一个元素的后缀为suff(i).把s的每个后缀按照字典序
排序，后缀数组sa[i]就表示排名为i的后缀的起始位置的下标,映射数组rk[i]就表示起始位置的下标为i的后缀的排名，sa表示排名为i的是啥，
rk表示第i个的排名是啥。
    最长公共前缀——后缀数组的辅助工具LCP: LCP(i,j)为suff(sa[i])与suff(sa[j])的最长公共前缀关于LCP的几条性质: 1、LCP(i,j)=LCP(j,i)
2、LCP(i,i)=len(sa[i])=n-sa[i]+1 3、LCP(i,k)=min(LCP(i,j),LCP(j,k)) 对于任意1<=i<=j<=k<=n
    4、LCP(i,k)=min(LCP(j,j-1))
对于任意1<i<=j<=k<=n
    如何求LCP?
    设height[i]为LCP(i,i-1)，1<i<=n，显然height[1]=0. 由LCP Theorem可得，LCP(i,k)=min(height[j]) i+1<=j<=k
设h[i]=height[rk[i]]，同样的，height[i]=h[sa[i]];  =>  h[i]>=h[i-1]-1;

应用:
1)求两个字符串最大的公共子串
    将两个字符串连接起来，中间加一个其它字符，这样只需求出设height,只需要扫一遍height即可得到最优结果。height[i]为LCP(i,i-1),
1<i<=n, LCP(i,j)为suff(sa[i])与suff(sa[j])的最长公共前缀.
for(int i=1;i<=n;i++){
    if(sa[i]<p&&sa[i-1]>p) res = max(res,height[i]);
    if(sa[i]>p&&sa[i-1]<p) res = max(res,height[i]);
}

2) 求一个串中所有子串的种类个数
    举例多个字符串，将其后缀排序后进行枚举发现,当前字符串的贡献为与下一个字符串的非公共长度，最后一个只需要全加即可。
而height[i]=LCP(i-1,i)的最长公共前缀。
ll res=0;
for(int i=2;i<=n;i++){
    int top = height[i];
    res+=(n-sa[i-1]+1-top);
}
res+=(n-sa[n]+1);

3) 求一个数组中数重复出现次数>=k的子序列(可重叠)最长长度是多少
    建立后缀数组,对于每个height[i]进行枚举,使用二分+st表查询以i为中心,[l,r]区间的最大值,其中[l,r]区间中每个值都>=height[i]
更新区间长度最大值即可.

4) 查询一个数组中满足模式相似的子序列(一个子序列加上一个值也算相似)重复出现2次以上,且没有交集的最大长度
    可以先进行差分,取得差值[1,n-1],然后对于差值建立后缀数组,然后通过二分查询值进行判断是否存在可行解.
bool isOK(int n,int k){
    vector<pii>v;
    for(int i=2;i<=n;i++){
        if(height[i]+1>=k){
            int j;
            for(j=i;j<=n;j++){
                if(height[j]+1<k) break;
                int l1=sa[j-1],r1=sa[j-1]+k-1;
                int l2=sa[j],r2=sa[j]+k-1;
                if(check(l1,r1,l2,r2)) return true;
                for(int k=0;k<v.size();k++){
                    if(check(l2,r2,v[k].first,v[k].second)) return true;
                }
```

```
                v.push_back(make_pair(l1,r1));
            }
            i=j; v.clear();
        }
    }
    return false;
}
```

5）给定一个字符串，求重复次数最多的连续重复子串,输出最大的重复次数

本题是一道裸的后缀数组题，"重复次数最多的连续重复子串"解法(摘自罗穗骞的国家集训队论文):先穷举长度L，然后求长度为L的子串最多能连续出现几次。首先连续出现1次是肯定可以的，所以这里只考虑至少2次的情况。假设在原字符串中连续出现2次，记这个子字符串为S，那么S肯定包括了字符r[0]，r[L]，r[L*2],r[L*3]，……中的某相邻的两个。所以只须看字符r[L*i]和r[L*(i+1)]往前和往后各能匹配到多远。最后看最大值是多少。

穷举长度L的时间是n，每次计算的时间是n/L。所以整个做法的时间复杂度是O(n/1+n/2+n/3+……+n/n)=O(nlogn)。

要提一提的总共有两点，第一点比较显而易见 "S肯定包括了字符r[0]，r[L]，r[L*2],r[L*3]，……中的某相邻的两个"由于当前S是有两个长度为L的连续重复子串拼接而成的，那意味着S[i]和S[i+L](0≤i<L)必定是一样的字符，而这两个字符位置相差L而字符r[0],r[L],r[L*2],r[L*3],.....中相邻两个的位置差均为L“只须看字符r[L*i]和r[L*(i+1)]往前和往后各能匹配到多远”，对于往后能匹配到多远，这个直接根据最长公共前缀就能很容易得到，即上图中的后缀Suffix(6)和后缀Suffix(9)的最长公共前缀。而对于往前能匹配到多远，我们当然可以一开始就把字符串反过来拼在后面，这样也能根据最长公共前缀来看往前能匹配到多远，但这样效率就比较低了。

其实，当枚举的重复子串长度为i时，我们在枚举r[i*j]和r[i*(j+1)]的过程中，必然可以出现r[i*j]在第一个重复子串里，而r[i*(j+1)]在第二个重复子串里的这种情况，如果此时r[i*j]是第一个重复子串的首字符，这样直接用公共前缀k除以i并向下取整就可以得到最后结果。但如果r[i*j]如果不是首字符，这样算完之后结果就有可能偏小，因为r[i*j]前面可能还有少许字符也能看作是第一个重复子串里的。于是，我们不妨先算一下，从r[i*j]开始，除匹配了k/i个重复子串，还剩余了几个字符，剩余的自然是k%i个字符。如果说r[i*j]的前面还有i-k%i个字符完成匹配的话，这样就相当于利用多余的字符还可以再匹配出一个重复子串，于是我们只要检查一下从r[i*j-(i-k%i)]和r[i*(j+1)-(i-k%i)]开始是否有i-k%i个字符能够完成匹配即可，也就是说去检查这两个后缀的最长公共前缀是否比i-k%i大即可。

当然如果公共前缀不比i-k%i小，自然就不比i小，因为后面的字符都是已经匹配上的，所以为了方便编写，程序里面就直接去看是否会比i小就可以了。

代码:

```
int calc(int i,int j){
    int x = rk[i];
    int y = rk[j];
    if(x>y) swap(x,y);
    return st.query_min(x+1,y);
}

int ans=1;
for(int i=1;i<n;i++){
    for(int j=1;j+i<=n;j+=i){
        int lcp = calc(j,j+i);
        int pos = j-(i-lcp%i);
        int res = lcp/i+1;
        if(pos>=1&&calc(pos,pos+i)>=i) res++;
```

```
94          ans=max(ans,res);
95      }
96  }
97  printf("%d\n",ans);
98
99  或者:
100 for(int i=1;i<=n;i++){
101     for(int j=1;j+i<=n;j+=i){
102         int len = st.query_min(rk[j],rk[j+i]);
103         int re = i-len%i;
104         if(re==i){
105             ans=max((len+i)/i,ans);
106         }else{
107             int p=j-re;
108             if(p>=1&&st.query_min(rk[p],rk[p+i])==len+re)
109                 ans=max(ans,(len+re+i)/i);
110         }
111     }
112 }
113
114 若输出循环次数最多的，最小字典序子串，那么可以存一个repeat，然后保存循环次数==repeat的长度，然后结合后
        缀数组性质暴力枚举即可。
115 int main() {
116     int Case=1;
117     while(scanf("%s",s+1)==1){
118         if(strcmp(s+1,"#")==0) break;
119         printf("Case %d: ",Case++);
120         n=strlen(s+1);
121         DA(s,n,128);st.init(n);
122         vector<int>v;
123         int repeat=0;
124         for(int i=1;i<=n;i++){
125             for(int j=1;j+i<=n;j+=i){
126                 int len = st.query_min(rk[j],rk[j+i]);
127                 int re=i-len%i,cnt=0;
128                 if(re==i) cnt=(i+len)/i;
129                 else{
130                     int p=j-re;
131                     if(p>=1&&st.query_min(rk[p],rk[p+i])>=re+len) cnt=(i+len+re)/i;
132                 }
133                 if(cnt>repeat){
134                     repeat=cnt; v.clear();v.push_back(i);
135                 }else if(cnt==repeat) v.push_back(i);
136             }
137         }
138         if(repeat==1){
139             char c = 'z';
140             for(int i=1;i<=n;i++) c=min(c,s[i]);
141             printf("%c\n",c);
142             continue;
143         }
144         int be=0,en=0;
145         //枚举
146         for(int i=1;i<=n&&!be;i++){
147             for(int j=0;j<v.size();j++){
148                 int p=sa[i],p2=p+v[j];
149                 if(p2>n) continue;
150                 int len = st.query_min(rk[p],rk[p2]);
151                 if(len>=(repeat-1)*v[j]){
```

```
                    be=p;en=p+repeat*v[j]-1;
                    break;
                }
            }
        }
        for(;be<=en;be++) printf("%c",s[be]);
        printf("\n");
    }
}
```

6）两个字符串，求三元组(i,j,k),即s[i]...s[i+k-1]和s[j]...s[j+k-1]相同的个数,k>=给定l。
计算A的某个后缀与B的某个后缀的最长公共前缀长度，如果长度L大于k，则加上L-k+1组。将两个字符串连接起来
，中间用一个没有出现的字符分开。（这是一个神奇的做法）然后通过height数组分组，某个组内的height都是大于等
于k的，也就是任意两个后缀的最长公共前缀都至少为k。扫描一遍，遇到一个B的后缀就与之前的A后缀进行统计，求出所有的满足的组数。但是这样的做法便是n^2的。可以发现两个后缀的最长公共前缀为这一段的height值的最小值。可以通过一个单调栈来维护一下，当前要入栈元素如果小于栈底元素，说明之后加入的B后缀与栈底的最长公共前缀是小
于等于入栈的。这样就保证了单调栈内的height值是绝对递增的，逐渐合并，均摊可以达到o(n)的复杂度。然后扫描两遍即可。

```
int main() {
    while(scanf("%d",&k)==1){
        if(!k) break;
        scanf("%s%s",s+1,s2+1);
        int len=strlen(s+1);
        s[0]=1;strcat(s,"#");strcat(s,s2+1);s[0]=0;
        int Len=strlen(s+1);
        DA(s,Len,128);
        ll ans=0,tot=0;
        int top=0;
        //a扫b
        for(int i=1;i<=Len;i++){
            if(height[i]<k) top=tot=0;
            else{
                ll cnt=0;
                if(sa[i-1]<=len) cnt++,tot+=height[i]-k+1;
                while(top>0&&st[top-1][0]>=height[i]){
                    top--;
                    tot-=st[top][1]*(st[top][0]-height[i]);
                    cnt+=st[top][1];
                }
                st[top][0]=height[i];st[top++][1]=cnt;
                if(sa[i]>len) ans+=tot;
            }
        }
        //b扫a
        for(int i=1;i<=Len;i++){
            if(height[i]<k) top=tot=0;
            else{
                ll cnt=0;
                if(sa[i-1]>len) cnt++,tot+=height[i]-k+1;
                while(top>0&&st[top-1][0]>=height[i]){
                    top--;
                    tot-=st[top][1]*(st[top][0]-height[i]);
                    cnt+=st[top][1];
                }
                st[top][0]=height[i];st[top++][1]=cnt;
```

```
                if(sa[i]<=len) ans+=tot;
            }
        }
        printf("%lld\n",ans);
    }
}
```

7）有n个串，求最长子串，它的子串或者反串都在n个字符串中出现过

只需要将正串和反串都加入后缀数组，然后计算即可。

```
    int t;
    scanf("%d",&t);
    while(t--){
        scanf("%d",&n);
        int left=1,right=100,len=1,sp=120;
        for(int i=1;i<=n;i++){
            scanf("%s",str);
            int l=strlen(str);
            right=min(right,l);
            for(int j=0;j<l;j++) {
                mark[len] =2*i-1; s[len++] = str[j]-'0'+1;
            }
            s[len++]=sp++;
            for(int j=l-1;j>=0;j--) {
                mark[len] =2*i; s[len++] = str[j]-'0'+1;
            }
            s[len++]=sp++;
        }
        len--;
        DA(s,len,sp++);
        while(left<=right){
            int mid=(left+right)>>1;
            if(isOK(mid,len,sp)) left=mid+1;
            else right=mid-1;
        }
        printf("%d\n",right);
    }
```

后缀自动机:

一个子串，它在原串中可能出现在若干的位置。而一个子串p出现的这些位置的右端点标号组成的集合，我们称之为 endpos(p)

1.如果两个子串的endpos相同，则其中子串一个必然为另一个的后缀

2.对于任意两个子串t和p(len_t<=len_p)，要么endpos(p)□ endpos(t),要么endpos(t) ∩ endpos(p)=□

3.对于endpos相同的子串，我们将它们归为一个endpos等价类。对于任意一个endpos等价类，将包含在其中的所有子串依长度从大到小排序，则每一个子串的长度均为上一个子串的长度减 1，且为上一个子串的后缀（简单来说，一个endpos等价类内的串的长度连续）

4.endpos等价类个数的级别为O(n)

5.一个类a中，有最长的子串，也有最短的子串，我们称最长子串的长度为len(a)，最短子串长度为minlen(a)。对于存在父子关系的两个类，设fa(a)表示类a的父亲（也是一个类）。则: len(fa(a))+1=minlen(a)

6.后缀自动机的边数为O(n)

后缀自动机的性质:

1.有一个源点，边代表在当前字符串后增加一个字符。

2.每个点代表一个endpos等价类，到达一个点的路径形成的子串必须属于此点的类。

3.点之间有父子关系，到达点i的所有字符串的长度都必然大于到达fa(i)的所有字符串的长度，且到达 fa(i)的任意一字符串必为到达i的任

意一字符串的后缀。
  4.每个节点都代表不同的endpos等价类,longest[i]-minlen[i]+1表示其中的字符串个数,每个节点所包含的字符串都不相同,且有满足
len(fa(a))+1=minlen(a),因此每个节点只需要记录最长的len[i]即可,最小值只需要找父节点来确定。
  5.数组要开两倍

学习文章: hihocoder里面关于后缀自动机讲解

应用:

1、求一个字符串中子串*子串个数的最大值,其中个数要大于1
代码:
```
void solve(){
    ll ans=0;
    for(int i=1;i<=tot;i++) c[len[i]]++;
    for(int i=1;i<=tot;i++) c[i]+=c[i-1];
    for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
    for(int i=tot;i>=1;i--){
        int p = rk[i];
        sz[fa[p]]+=sz[p];
        if(sz[p]>1)
            ans=max(ans,(ll)sz[p]*len[p]);
    }
    printf("%lld\n",ans);
}
```

2、求一个字符串中长度大于等于m的个数总和
```
void solve(){
    ll ans=0;
    for(int i=tot;i>=1;i--){
        if(len[i]<m)
            continue;
        else{
            ans+=(len[i]-max(m,len[fa[i]]+1)+1);
        }
    }
    printf("%lld\n",ans);
}
```

3、求两个字符串的最长公共子串
方法1:建立两个后缀自动机,然后dfs(1,1,0),第一个参数表示sam1的节点,第二个参数是sam2的节点,第三个参数是公共长度
方法2:建立一个后缀自动机,然后将另外一个字符串进行沿点搜索
```
void query(char *s){
    int ret=0,p=1,nowlen=0;
    int n=strlen(s);
    for(int i=0;i<n;i++){
        int ch=s[i]-'a';
        if(nx[p][ch]){p=nx[p][ch];ret=max(ret,++nowlen);continue;}
        //如果没有匹配成功则p=fa[p],保证endpos一样, 有满足min{p}=max{fa[p]}+1
        //因为p节点都是fa[p]前面加一个字母而来的,所以回去父节点保证了以i为点的匹配最长
        while(p&&!nx[p][ch]) p=fa[p];
        if(!p) nowlen=0,p=1;
        else{
            //注意不可以p=nx[p][c]; nowlen=len[fa[p]]+1; 因为fa和nx数组并不是相互映射的关系
            ret=max(ret,nowlen=len[p]+1); p=nx[p][ch];
        }
    }
        printf("%d\n",ret);
```

```
}

4、给定两个字符串，求出在两个字符串中各取出一个子串使得这两个子串相同的方案数。两个方案不同当且仅当这两个子串中有一个位置不同。
方法一：先建立对一串建立后缀自动机,然后操作如下：
inline void topsort(){
    for(int i=1;i<=tot;i++) c[len[i]]++;
    for(int i=1;i<=tot;i++) c[i]+=c[i-1];
    for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
    for(int i=tot;i>=1;i--){int p=rk[i]; sz[fa[p]]+=sz[p];}
    //表示状态转移到p节点时，dp[p]能新增多少答案
    for(int i=1;i<=tot;i++){
        int p=rk[i];dp[p]=dp[fa[p]]+sz[p]*(len[p]-len[fa[p]]);
    }
}

inline void solve(char *s){
    ll ret=0;
    int p=1,nowlen=0;
    int n=strlen(s);
    for(int i=0;i<n;i++){
        int c=s[i]-'a';
        if(nx[p][c]){
            p=nx[p][c]; nowlen++; ret+=dp[fa[p]]+sz[p]*(nowlen-len[fa[p]]);
            continue;
        }
        for(;p&&!nx[p][c];p=fa[p]);
        if(!p) p=1,nowlen=0;
        else{
            nowlen=len[p]+1;
            p=nx[p][c];
            ret+=dp[fa[p]]+sz[p]*(nowlen-len[fa[p]]);
        }
    }
    printf("%lld\n",ret);
}

5、给定一个字符串,现在想要构造此字符串,增加任意一个字符串,需要花费p元,从之前任意已经生成子串中增加为q,求最小花费
    对于i从小到大处理，维护使得s[j:i]□ s[1:j-1]的最小的j，那么记f[i]为输出前i个字符的最小代价，则f[i]=min{f[i-1]+p,f[j-1]+q}。
用SAM维护s[1:j-1]，若s[1:j-1]中包含s[j:i+1]，即加入第 i + 1 个字符仍然能复制，就不需要做任何处理。否则，重复地将第 j 个字符加入
后缀自动机并j=j+1，相应维护s[j:i+1]在后缀自动机上新的匹配位置，直到s[j,i + 1]□ s[1,j -1]。
class SuffixAutoMaton{
public:
    int last,tot,p;
    int nx[N<<1][dif],fa[N<<1],len[N<<1];
    void init(){
        last=tot=1; p=1;
        fa[1]=len[1]=0;
        memset(nx[1],0, sizeof(nx[1]));
    }

    inline int match(char ch){
        return nx[p][ch-'a'];
    }

    inline void withdraw(int l){
```

```
        while(p!=0&&len[fa[p]]>=l) p=fa[p];
        if(p==0) p=1;
    }

    void transfer(int l,int ch){
        p=nx[p][ch];
        if(p==0) p=1;
        withdraw(l);
    }

    inline void insert(int c){
        int p=last,np=++tot;
        memset(nx[tot],0, sizeof(nx[tot]));
        last=np; len[np]=len[p]+1;
        for(;p&&!nx[p][c];p=fa[p]) nx[p][c]=np;
        if(!p) fa[np]=1;
        else{
            int q=nx[p][c];
            if(len[p]+1==len[q]) fa[np]=q;
            else{
                int nq=++tot;
                len[nq]=len[p]+1;
                memcpy(nx[nq],nx[q], sizeof(nx[q]));
                fa[nq]=fa[q]; fa[q]=fa[np]=nq;
                for(;nx[p][c]==q;p=fa[p])
                    nx[p][c]=nq;
            }
        }
    }

}SAM;


void solve(){
    SAM.init();
    SAM.insert(s[0]-'a');
    dp[0]=p;
    int l=1,r=0;
    for(int i=1;i<n;i++){
        ++r;
        dp[i]=dp[i-1]+p;
        while( ( !SAM.match(s[i]) || r-l+1>(i+1)/2 ) && l<=r ){
            SAM.insert(s[l++]-'a');
            SAM.withdraw(r-l);
        }
        SAM.transfer(r-l+1,s[i]-'a');
        if(l<=r){
            dp[i]=min(dp[i],dp[i-(r-l+1)]+q);
        }
    }
    printf("%lld\n",dp[n-1]);
}
```

6、小Hi发现旋律可以循环，每次把一段旋律里面最前面一个音换到最后面就成为了原旋律的“循环相似旋律”，还可以对“循环相似旋律”进
行相同的变换能继续得到原串的“循环相似旋律”。小Hi对此产生了浓厚的兴趣，他有若干段旋律，和一部音乐作品。对于每一段旋律，他想
知道有多少在音乐作品中的子串（重复便多次计）和该旋律是“循环相似旋律”。

输入:
第一行，一个由小写字母构成的字符串S，表示一部音乐作品。字符串S长度不超过100000。第二行，一个整数N，表示有N段旋律。接下来N行，
每行包含一个由小写字母构成的字符串str，表示一段旋律。所有旋律的长度和不超过100000。

abac
3
a
ab
ca

输出:
输出共N行，每行一个整数，表示答案。
2
2
1

```
//将字符串s->s+s,然后进入处理
void query(char *s){
    int n=strlen(s),p=1,nowlen=0,limit=n/2;
    memset(vis,0, sizeof(bool)*(tot+1));
    ll ret=0;
    for(int i=0;i<n-1;i++){
        int c=s[i]-'a';
        if(nowlen==limit&&nx[p][c]){
            nowlen++;p=nx[p][c];
            if(nowlen>len[fa[p]]+1) nowlen--;
            else{
                nowlen--; p=fa[p];
            }
        }else{
            if(nx[p][c]){
                nowlen++;p=nx[p][c];
            }else {
                while (p && !nx[p][c]) p = fa[p];
                if (!p) {
                    p = 1;nowlen = 0;
                } else {
                    nowlen = len[p] + 1;p = nx[p][c];
                }
            }
        }
        if(nowlen==limit&&!vis[p]){
            ret+=sz[p];vis[p]=1;
        }
    }
    printf("%lld\n",ret);
 }
```

7、对于一个给定长度为N的字符串，求它的第K小子串是什么。
两个整数T和K，T为0则表示不同位置的相同子串算作一个。T=1则表示不同位置的相同子串算作多个。

思路:
    建立后缀自动机，然后进行记忆化搜索，用dp[i]表示以i为节点的子树所包含的子串个数。

代码:

```
//先记忆化搜索，得到以u为根节点的子树所包含的子串数
void dfs(int u){
    dp[u]=sz[u];
```

```
    for(int i=0;i<26;i++){
        int v=nx[u][i];
        if(!v) continue;
        if(!dp[v]) dfs(v);
        dp[u]+=dp[v];
    }
}

//对答案进行计算
void DFS(int u,ll k){
    if(sz[u]>=k) return;
    k-=sz[u];
    for(int i=0;i<26;i++){
        int v=nx[u][i];
        if(!v) continue;
        if(dp[v]>=k){
            printf("%c",i+'a');
            DFS(v,k); return;
        }else k-=dp[v];
    }
}

void query(int t,ll k){
    topu(t); sz[1]=0; dfs(1);
    if(dp[1]<k){
        printf("-1\n");return;
    }
    DFS(1,k);
    printf("\n");
}
```

8、给出n个串，求这个n个串的最长公共子串，1≤n≤10,|s|≤1e5

对第一个串建立一个后缀自动机，然后用剩余n-1个串进行匹配，设mlen[i]表示i节点，n-1个串匹配长度的最大值中的最小值

那么最后答案就是遍历所有节点的mlen,最大值即为答案。其中需要注意的是如果i节点匹配成功，那么其父节点的最大值为其父节点长度

代码:

```
bool Update(){
    if(scanf("%s",s)!=1) return 0;
    memset(clen,0, sizeof(int)*(tot+1));
    int n=strlen(s),nowlen=0,p=1;
    //进行节点匹配，clen[i]表示当前节点匹配的最大值
    for(int i=0;i<n;i++){
        int c=s[i]-'a';
        if(nx[p][c]){
            p=nx[p][c];nowlen++;
        }else{
            while(p&&!nx[p][c]) p=fa[p];
            if(!p){
                p=1;nowlen=0;
            }else{
                nowlen=len[p]+1;p=nx[p][c];
            }
        }
        clen[p]=max(clen[p],nowlen);
    }
    //更新mlen[i],并更新父节点的clen
    for(int i=tot;i>=1;i--){
```

```
        int p=rk[i];
        mlen[p]=min(mlen[p],clen[p]);
        if(clen[p]&&fa[p]) clen[fa[p]]=len[fa[p]];
    }
    return 1;
}
```

9、给定一个长度为n的字符串s，令Ti表示它从第i个字符开始的后缀，求sigma(1<=i<j<=n)len[i]+len[j]-2*lcp(Ti,Tj)的和
其中,len[i]表示字符串i的长度,lcp(a,b)表示字符串a和字符串b的最长公共前缀

思路:
首先把字符串反过来，前缀变后缀，然后建立后缀自动机。我们发现parent树即后缀树，它是不断在前面加字符串而导致endpos集
分裂，假设一个跟节点为z，两个子节点为x,y，那么两者lca的节点表示的长度是他们的最长匹配子串,因此我们只需要对每个节点进行
计算即可。

代码:

```
void query(ll n){
    ll ret=(n-1)*n*(n+1)/2;
    for(int i=1;i<=tot;i++) c[len[i]]++;
    for(int i=1;i<=tot;i++) c[i]+=c[i-1];
    for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
    for(int i=tot;i>=1;i--){
        int p=rk[i]; sz[fa[p]]+=sz[p];
        ret-=(len[p]-len[fa[p]])*sz[p]*(sz[p]-1);
    }
    printf("%lld\n",ret);
}
```

广义后缀自动机:

1、神奇的是小Hi发现了一部名字叫《十进制进行曲大全》的作品集，顾名思义，这部作品集里有许多作品，但是所有的作品有一个共同特征:
只用了十个音符，所有的音符都表示成0-9的数字。现在小Hi想知道这部作品中所有不同的旋律的“和”（也就是把串看成数字，在十进制下的求和，
允许有前导0）。答案有可能很大，我们需要对（10^9 + 7)取模。

输入:

```
2
101
09
```

输出:

```
131
```

思路:
    建立广义后缀自动机，然后进行拓扑排序，累计答案即可。

```
ll dp[N<<1],cnt[N<<1];
int rk[N<<1],c[N<<1];

void query(){
    for(int i=1;i<=tot;i++) c[len[i]]++;
    for(int i=1;i<=tot;i++) c[i]+=c[i-1];
    for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
    cnt[1]=1;
    //正向扫可以得到以1节点拓扑排序
```

```
    for(int i=1;i<=tot;i++){
        int u=rk[i];
        for(int j=0;j<10;j++){
            int v=nx[u][j];
            if(!v) continue;
            cnt[v]=(cnt[v]+cnt[u])%mod;
            dp[v]=(dp[v]+10*dp[u]%mod+cnt[u]*j%mod)%mod;
        }
    }
    ll ret=0;
    for(int i=2;i<=tot;i++) ret=(ret+dp[i])%mod;
    printf("%lld\n",ret);
}
```

7.9 回文自动机

Palindromic Tree，译名为“回文树”，是一种专门处理回文串的数据结构，类似于Manachar算法，但更为强大。是由两颗分别存储偶数回文
串树和存储奇数回文串树组成，每个节点代表母串的回文串，两树之间中用fail指针连接。

假设我们有一个串S，S下标从0开始，则回文树能做到如下几点：
1.求串S前缀0~i内本质不同回文串的个数（两个串长度不同或者长度相同且至少有一个字符不同便是本质不同）
2.求串S内每一个本质不同回文串出现的次数
3.求串S内回文串的个数（其实就是1和2结合起来）
4.求以下标i结尾的回文串的个数

应用：

1、求最长回文子串
回文自动机中节点长点最长的即是答案

2、求字符串中本质不同回文子串的数量
回文自动机中除了0、1节点所产生的节点数量即为本质不同回文子串数量。

3、求两个字符串有多少回文子串能配对的数量
For example, (1, 3, 1, 3) and (1, 3, 3, 5) are both considered as a valid common palindrome substring
between “aba” and “ababa”. aba能匹配两次.
建成两颗回文树后,只需要dfs(0,0)偶子树和dfs(1,1)奇子树,只要遍历到两个点同时存在,则数量ans += 1ll*pt1.cnt[to1]*pt2.cnt[to2];

4、求一个字符串求最长双回文子串T,即可将T分为两部分X,Y,(|X|,|Y|≥1)且X和Y都是回文串。
只需要前后跑一次回文自动机,l[i]表示从左到i所形成的最长回文串,r[i]表示从i到右所形成的的最长回文串,遍历一遍字符串即可得到答案。

5、求一个字符串每个前缀中有多少个本质不同的回文串
每个字母插入时,统计一遍现在回文自动机中所产生的节点总数即可.

6、求一个字符串(1≤T≤10, 1≤length≤1000, 1≤Q≤100000, 1≤l≤r≤length)，区间[l,r]中本质不同的回文串数
考虑到1≤length≤1000,我们可以强行打表记录ans[i][j],表示[i,j]之间回文子串个数,而不是在Q中每次查询情况.

7、定义合法(x,y)为两个不相交的回文串,求个数.
解释: aca, S1=T[0,0],S2=T[0,2],S3=T[1,1],S4=T[2,2],其中(S1,S3) (S1,S4) (S3,S4)为合法对
从回文自动机的fail边可知: 边是连接一个字符串与另一个字符串的最长后缀回文子串,因此可以查询一个点到0点长度即加入一个点后
所增加的回文串,再反着打一次后缀和,结果相乘即是答案。

```
36
37  int getsum(int from){
38      int res=1,x=from;
39      if(vis[x])
40          return vis[x];
41      if(pt.fail[from]!=0&&pt.fail[from]!=1){
42          int to=pt.fail[x];
43          res+=vis[to];
44      }
45      vis[from]=res;
46      return vis[from];
47  }
48
49  int main() {
50      while(scanf("%s",str)==1) {
51          int len = strlen(str);
52          pt.init();
53          memset(vis,0, sizeof(int)*(len+1));
54          for(int i=0;i<len;i++){
55              pt.insert(str[i]-'a');
56              l[i]=getsum(pt.last);
57          }
58          memset(vis,0, sizeof(int)*(len+1));
59          pt.init();
60          r[len]=0;
61          for(int i=len-1;i>=1;i--){
62              pt.insert(str[i]-'a');
63              r[i]=r[i+1]+getsum(pt.last);
64          }
65          ll ans=0;
66          for(int i=0;i<len-1;i++){
67              ans+=l[i]*r[i+1];
68          }
69          printf("%lld\n",ans);
70      }
71      return 0;
72  }
73
74  8、有奇数个，并且他们手中的牌子所写的字母，从左到右和从右到左读起来一样，那么这一段女生就被称作和谐小群
        体。
75  现在想找出所有和谐小群体，并且按照女生的个数降序排序之后，前K个和谐小群体的女生个数的乘积是多少。
76      显然回文自动机上节点的标记就是他们出现的时间,以及len可以记录,因此只需要dfs(1)一遍奇回文树,即可得
        到每个节点的
77  长度和该节点表示回文串出现的次数,然后排序扫一遍即可。
78
79  9、three tuple (i,j,k) satisfy 1≤i≤j<k≤length(S), S[i..j] and S[j+1..k] are all
        palindrome strings.
80  wants to know the sum of i*k of all required three tuples. The answer may be very large
        , please output
81  the answer mod 1000000007.
82      设left[i]为以i为点从右往左的sigma(j)的和,right[i]以i为点从左往右的sigma(k)的和,因此需要dfs辅
        助记录点数和长度
83  然后sigma(left[i]*right[i+1])即是答案.
84
85  void Update(int from){
86      int res=1,res2=pt.len[from];
87      if(vis[from])
88          return;
89      if(pt.fail[from]!=0&&pt.fail[from]!=1){
```

```
        int to = pt.fail[from];
        res+=vis[to];
        res2+=ls[to];
        res%=mod;
        res2%=mod;
    }
    vis[from]=res;
    ls[from]=res2;
    return;
}

9、实现可前后插入字符,查询当前有多少个本质不同回文串,已经生成回文串个数
    给你n次操作，如果为1，则在字符串后面插入一个字符，如果为2，则在字符串前面插入一个字符，如果为3，则输出当前的字符
串中的本质不同的回文串的个数，如果为4，则输出字符串的回文串的个数。

typedef pair<int, int> pii;
typedef long long ll;
const double eps = 1e-6;
const ll INF = 0x3f3f3f3f3f3f3f3f;
const ll mod = 1000000007;
const int N = 2e5 + 10;
const int M = 2e5 + 10;

const double PI = acos(-1.0);

const int dif = 26;
int n, q;
int op;
char str[5];


class PalindromicTree{
public:
    int nx[N][dif],fail[N],len[N],num[N];
    int tot[2],p,last[2],s[N];
    int newnode(int l){
        memset(nx[p],0, sizeof(nx[p]));
        len[p]=l;
        num[p]=0;
        return p++;
    }
    void init(int x){
        last[0]=last[1]=p=0;
        tot[0]=x;tot[1]=x-1;
        fail[0]=fail[1]=1;
        memset(s,-1, sizeof(s));
        newnode(0);
        newnode(-1);
    }
    int getfail(int x,int tag){
        if(!tag){
            while(s[tot[tag]+len[x]+1]!=s[tot[tag]])
                x=fail[x];
        }else{
            while(s[tot[tag]-len[x]-1]!=s[tot[tag]])
                x=fail[x];
        }
        return x;
```

```
    }
    int insert(int x,int tag){
        if(!tag)
            s[--tot[0]]=x;
        else
            s[++tot[1]]=x;
        int cur = getfail(last[tag],tag);
        int now = nx[cur][x];
        if(!now){
            now = newnode(len[cur]+2);
            fail[now]=nx[getfail(fail[cur],tag)][x];
            nx[cur][x]=now;
            num[now]+=num[fail[now]]+1;
        }
        last[tag]=nx[cur][x];
        if(len[last[tag]]==tot[1]-tot[0]+1)
            last[tag^1]=last[tag];
        return num[last[tag]];
    }
}pt;


int main() {
    while(scanf("%d",&q)==1){
        pt.init(q);
        ll ans=0;
        while(q--){
            scanf("%d",&op);
            if(op==1){
                scanf("%s",str);
                ans+=pt.insert(str[0]-'a',0);
            }else if(op==2){
                scanf("%s",str);
                ans+=pt.insert(str[0]-'a',1);
            }else if(op==3){
                printf("%d\n",pt.p-2);
            }else{
                printf("%lld\n",ans);
            }
        }
    }
    return 0;
}
```

10、两个相交的回文串为一对,求一个字符串中有多少对。
直接求相交非常麻烦,且非常难处理,可以换一个思路,先求总和,以及不会相交的，那么相减即可得到结果。但是内存有限定,因此可以使用
vector进行优化。

```
class PalindromicTree{
public:
    vector<pii>nx[N];
    int fail[N],len[N],num[N];
    int tot,p,last,s[N];
    int newnode(int l){
        nx[p].clear();
        len[p]=l;
        num[p]=0;
        return p++;
    }
```

```
    void init(){
        tot=p=last=0;
        s[0]=-1,fail[0]=1;
        newnode(0);
        newnode(-1);
    }
    int getfail(int x){
        while(s[tot-len[x]-1]!=s[tot])
            x=fail[x];
        return x;
    }

    int is_exist(int p,int c){
        for(auto t:nx[p]){
            if(t.first==c)
                return t.second;
        }
        return 0;
    }

    void insert(int x){
        s[++tot]=x;
        int cur = getfail(last);
        int now = is_exist(cur,x);
        if(!now){
            now = newnode(len[cur]+2);
            fail[now]=is_exist(getfail(fail[cur]),x);
            nx[cur].push_back(make_pair(x,now));
            num[now]=num[fail[now]]+1;
        }
        last=now;
    }
}pt;

11、求一个字符串中有多少子串满足,1.r-l+1==i 2.子串为回文串 3.[l,(l+r)/2]也为回文串
    有可能为答案的是回文自动机上的节点,只需要节点判断是否满足即可,关键在于判断[l,(l+r)/2]也为回文串,
    可以发现若满足条件三
则前半段和后半段是一样的,因此可以用hash判断前半段和后半段是否相同即可，如果相同则进入计数。
const ull hash1 = 201326611;
const ull hash2 = 50331653;
ull ha[N],pp[N];

ull getha(int l,int r){
    if(l==0)
        return ha[r];
    return ha[r]-ha[l-1]*pp[r-l+1];
}

bool check(int l,int r){
    int len = r-l+1;
    int mid = (l+r)>>1;
    if(len&1)
        return getha(l,mid)==getha(mid,r);
    else
        return getha(l,mid)==getha(mid+1,r);
}

class PalindromicTree{
public:
```

```
264     int nx[N][dif],fail[N],len[N],cnt[N];
265     int tot,p,last,s[N],id[N];
266     int newnode(int l){
267         memset(nx[p],0, sizeof(nx[p]));
268         len[p]=l;
269         cnt[p]=0;
270         return p++;
271     }
272     void init(){
273         tot=p=last=0;
274         s[0]=-1,fail[0]=1;
275         newnode(0);
276         newnode(-1);
277     }
278     int getfail(int x){
279         while(s[tot-len[x]-1]!=s[tot])
280             x=fail[x];
281         return x;
282     }
283     void insert(int x){
284         s[++tot]=x;
285         int cur = getfail(last);
286         int now = nx[cur][x];
287         if(!now){
288             now = newnode(len[cur]+2);
289             fail[now]=nx[getfail(fail[cur])][x];
290             nx[cur][x]=now;
291         }
292         last=nx[cur][x];
293         cnt[last]++;
294         id[last]=tot;
295     }
296     void makecnt(){
297         for(int i=p-1;i>=2;i--)
298             cnt[fail[i]]+=cnt[i];
299         for(int i=2;i<p;i++){
300             if(check(id[i]-len[i],id[i]-1)){
301                 ans[len[i]]+=cnt[i];
302             }
303         }
304     }
305 }pt;
306
307 int main(){
308     pp[0]=1;
309     for(int i=1;i<N;i++)
310         pp[i]=hash1*pp[i-1];
311     while(scanf("%s",str)==1){
312         int len = strlen(str);
313         memset(ans,0, sizeof(int)*(len+2));
314         pt.init();
315         ha[0]=str[0];
316         for(int i=0;i<len;i++)
317             pt.insert(str[i]-'a');
318         for(int i=1;i<len;i++)
319             ha[i]=ha[i-1]*hash1+str[i];
320         pt.makecnt();
321         printf("%d",ans[1]);
322         for(int i=2;i<=len;i++)
```

```
            printf(" %d",ans[i]);
        printf("\n");
    }
}

12、求一个字符串中所有回文串中,若一个回文串包含另一个回文串,则为为一对,求所有满足情况的对
    先建立回文树,显然每个节点代表一种回文串,我们可以通过dfs进行求取.
void dfs(int x,ll res){
    vector<int>v;
    for(int i=x;i>1;i=fail[i]){
        if(!vis[i]){
            v.push_back(i);
            vis[i]=1;
            res++;
        }else break;
    }
    ans+=res;
    for(int i=0;i<26;i++) if(nx[x][i]) dfs(nx[x][i],res);
    for(auto t:v) vis[t]=0;
}

void solve(){
    ans=2-p;
    dfs(0,0);
    dfs(1,0);
    printf("%lld\n",ans);
}
```

7.10 线段树维护 dp

```
1.Wi-Fi(http://codeforces.com/contest/1216/problem/F)
题意:
    现在有n(1<=n<=2e5)个房间，要将每个房间通上网络，其中标号为1的房间可以放置路由器，可以使得[max(1,
    i-k),min(n,i+k)]
房间连上网络,标号为0的房间只能独自连上网络,每个房间连上网络的费用为i,即房间号1~n.现在求使得n个房间全部
    联通的最小花费

思路:
    设dp[i]表示1~i房间连上网的最小费用,当s[i]='0'时,则有dp[i]=min{dp[i-1]+i,dp[i]},当s[i]='1'时
    ,则有dp[i]=min{dp[i],dp[i-1+i],
dp[max{1,i-k}-1]+i]},并且dp[i]可以更新[max(1,i-k),min(n,i+k)]的所有值,因此可以使用线段树进行维
    护得到最终答案.
```