



---

ACM

---

**zjgsu**

Mr.l

February 24, 2020

# Contents

<b>0</b>	<b>头文件</b>	<b>1</b>
0.1	头文件 . . . . .	1
0.2	读入模板 . . . . .	1
<b>1</b>	<b>字符串</b>	<b>5</b>
1.1	KMP、E-KMP、Manacher . . . . .	5
1.2	字典树 . . . . .	6
1.3	哈希 . . . . .	7
1.4	AC 自动机 . . . . .	8
1.5	回文自动机 . . . . .	17
1.6	后缀数组、后缀自动机 . . . . .	20
<b>2</b>	<b>动态规划</b>	<b>28</b>
2.1	背包 dp . . . . .	28
2.2	数位 dp . . . . .	28
2.3	dp 优化 . . . . .	29
2.4	编辑距离 . . . . .	30
2.5	状压 dp . . . . .	30
<b>3</b>	<b>数据结构</b>	<b>32</b>
3.1	并查集 . . . . .	32
3.2	线段树 . . . . .	32
3.3	st 表 . . . . .	33
3.4	线性基 . . . . .	34
3.5	树分治 . . . . .	35
3.6	树重心 . . . . .	36
3.7	树状数组 . . . . .	37
3.8	树的直径 . . . . .	38
<b>4</b>	<b>图论</b>	<b>39</b>
4.1	最短路 . . . . .	39
4.2	最小生成树 . . . . .	42
4.3	强联通分量 . . . . .	43
4.4	网络流 . . . . .	45
4.5	最小树形图 . . . . .	47
4.6	割点、桥、双联通分量 . . . . .	50
4.7	二分匹配 . . . . .	56
4.8	第 k 最短路 . . . . .	57
4.9	2-SAT . . . . .	58
4.10	LCA . . . . .	60
4.11	欧拉路 . . . . .	64
<b>5</b>	<b>数学</b>	<b>67</b>
5.1	BM . . . . .	67
5.2	gcd、ex-gcd . . . . .	68
5.3	拉格朗日插值法 . . . . .	69
5.4	素数 . . . . .	71
5.5	高斯消元 . . . . .	75
5.6	几何基础模板 . . . . .	77
5.7	大数模板 . . . . .	79
5.8	组合数学 . . . . .	82
5.9	快速阶乘 . . . . .	83
5.10	FFT . . . . .	85
5.11	平面最近点对 . . . . .	87

<b>6</b>	<b>杂七杂八</b>	<b>89</b>
6.1	二分、三分查找	89
6.2	离散化	91
6.3	斯坦纳树	91
6.4	子矩阵问题	92
6.5	矩阵模板	95
6.6	汉诺塔	96
6.7	前缀和与差分	97
6.8	stl	100
6.9	最长上升子序列	102
<b>7</b>	<b>专题训练</b>	<b>105</b>
7.1	区间 dp	105
7.2	一般 dp	108
7.3	数位 dp	112
7.4	概率 dp	118
7.5	斜率 dp	122
7.6	树形 dp	125
7.7	KMP、E-KMP、Manacher	131
7.8	后缀数组、后缀树、后缀自动机	133
7.9	回文自动机	144
7.10	线段树维护 dp	150

## 0 头文件

### 0.1 头文件

```

1  #pragma comment(linker, "/STACK:1024000000,1024000000")
2  #include<iostream>
3  #include<cstdio>
4  #include<cmath>
5  #include<cstring>
6  #include<vector>
7  #include<algorithm>
8  #include<sstream>
9  #include<map>
10 #include<queue>
11 #include<set>
12 #include<bitset>
13 #include<list>
14 using namespace std;
15
16 typedef pair<int, int> pii;
17 typedef long long ll;
18 typedef unsigned long long ull;
19 #define pw(k) ((1ll)<<(k))
20 const ull hash1 = 201326611;
21 const double eps = 1e-8;
22 const ll INF = 0x3f3f3f3f3f3f3f3f;
23 const int inf = 0x3f3f3f3f;
24 const ll mod = 1e9 + 7;
25 const int N = 2e6+10;
26 const int M = 12;
27 const int dif = 26;
28 const double PI = acos(-1.0);
29 ll Mod(ll x){ return (x%mod+mod)%mod;}
30 void BinaryBitset(int n) { cout << bitset<sizeof(int) * 4>(n) << endl; }
31 inline int getBinary(int x){int cnt=0; for(;x;x=(x & (-x))) cnt++;return cnt;}
32
33 int main() {
34 #ifdef ACM_LOCAL
35     freopen("./std.in", "r", stdin);
36     //freopen("./std.out", "w", stdout);
37     auto start = clock();
38 #endif
39
40 #ifdef ACM_LOCAL
41     auto end = clock();
42     cerr << "Run Time: " << double(end - start) / CLOCKS_PER_SEC << "s" << endl;
43 #endif
44 }

```

### 0.2 读入模板

```

1  1、快读
2  struct ioss
3  {
4  #define endl '\n'
5      static const int LEN = 20000000;
6      char obuf[LEN], *oh = obuf;
7      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 ios& operator<<(string &c)
76     {
77         for (auto &p : c)
78             this->operator<<(p);
79         return *this;
80     }
81     inline ios& operator<<(const char *c)
82     {
83         while (*c != '\0')
84         {
85             this->operator<<(*c);
86             c++;
87         }
88         return *this;
89     }
90     inline ios& 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 ios& 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 ios& 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     ~ios()
123     {
124         fb->sputn(obuf, oh - obuf);

```

```

125     }
126 } io;
127
128 2、__int128读入
129
130 inline __int128 read(){
131     int X=0,w=0; char ch=0;
132     while(!isdigit(ch)) {wl=ch=='-';ch=getchar();}
133     while(isdigit(ch)) X=(X<<3)+(X<<1)+(ch^48),ch=getchar();
134     return w?-X:X;
135 }
136
137 void print(__int128 x){
138     if (!x) return ;
139     if (x < 0) putchar('-'),x = -x;
140     print(x / 10);
141     putchar(x % 10 + '0');
142 }
143
144 3、
145 template<class T>inline void read(T &res){
146     char c;T flag=1;
147     while((c=getchar())<'0' || c>'9')if(c=='-')flag=-1;res=c-'0';
148     while((c=getchar())>='0'&&c<='9')res=res*10+c-'0';res*=flag;
149 }
150
151
152 4、整行读入，其中包含空格
153 //注意前后使用getchar()清除多余空字符
154 int i=0;
155 while((s[i]=getchar())!='\n') i++;
156 s[i]=0;
157
158
159 //注意前后使用cin.get()清除多余空字符
160 getline(cin,s);
161
162 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
163 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 字符串

## 1.1 KMP、E-KMP、Manacher

```

1 1.KMP模板
2
3 //求取循环节的基础
4 void getNext(){
5     int i,j;
6     j=nx[0]=-1;
7     i=0;
8     while(i<n){
9         while(-1!=j && x[i]!=x[j]) j=nx[j];
10        nx[++i]=++j;
11    }
12 }
13
14 //消除循环节
15 void getNext(){
16     int i,j;
17     j=nx[0]=-1;
18     i=0;
19     while(i<m){
20         while(-1!=j && x[i]!=x[j]) j=nx[j];
21         if(x[++i]==x[++j]) nx[i] = nx[j];
22         else nx[i]=j;
23     }
24 }
25
26 //返回x在y中匹配次数, 包含重叠
27 int KMP() {
28     int i, j;
29     int ans = 0;
30     getNext(x, m, nx);
31     i = j = 0;
32     while (j < n) {
33         while (i != -1 && x[i] != y[j])
34             i = nx[i];
35         i++; j++;
36         if (i == m)
37             ans++;
38     }
39     return ans;
40 }
41
42 2、扩展KMP
43 void pre_EKMP(){
44     nx[0]=m;
45     int j = 0;
46     while(j+1<m&&x[j]==x[j+1]) j++;
47     nx[1]=j;
48     int k = 1;
49     for(int i=2;i<m;i++){
50         int p = nx[k]+k-1;
51         int L = nx[i-k];
52         if(i+L<p+1) nx[i]=L;
53         else{
54             j = max(0,p-i+1);
55             while(i+j<m&&x[i+j]==x[j]) j++;

```



```

56         nx[i]=j;
57         k=i;
58     }
59 }
60 }
61
62 void EKMP(){
63     pre_EKMP();
64     int j = 0;
65     while(j<n&&j<m&&x[j]==y[j]) j++;
66     extend[0]=j;
67     int k = 0;
68     for(int i=1;i<n;i++){
69         int p = extend[k]+k-1;
70         int L = nx[i-k];
71         if(i+L<p+1) extend[i]=L;
72         else{
73             j = max(0,p-i+1);
74             while(i+j<n&&j<m&&y[i+j]==x[j]) j++;
75             extend[i]=j;
76             k=i;
77         }
78     }
79 }
80
81 3、Manacher
82 //内存开两倍
83 const int N = 110100;
84 char Ma[N];
85 int Mp[N],top;
86 void Manacher(char *s,int len){
87     top=0;
88     Ma[top++]='$';
89     Ma[top++]='#';
90     for(int i=0;i<len;i++){
91         Ma[top++]=s[i];
92         Ma[top++]='#';
93     }
94     Ma[top]=0; Mp[top]=0;
95     int id=0,mx=0;
96     for(int i=1;i<top;i++){
97         Mp[i]=mx>i?min(Mp[2*id-i],mx-i):1;
98         while(Ma[i+Mp[i]]==Ma[i-Mp[i]]) Mp[i]++;
99         if(i+Mp[i]>mx){
100             mx=i+Mp[i];
101             id=i;
102         }
103     }
104 }

```

## 1.2 字典树

```

1 class tire{
2 public:
3     int nx[N][dif],end[N];
4     int root,tot;
5
6     int newNode(){

```

```

7         for(int i=0;i<26;i++) nx[tot][i]=-1;
8         end[tot]=0;
9         return tot++;
10    }
11
12    void init(){
13        tot=0;
14        root=newNode();
15    }
16
17    void insert(char *s,int id){
18        int len=strlen(s);
19        int now=root;
20        for(int i=0;i<len;i++){
21            int c=s[i]-'a';
22            if(nx[now][c]==-1) nx[now][c]=newNode();
23            now=nx[now][c];
24        }
25        end[now]=id;
26    }
27
28    int query(char *s){
29        int len=strlen(s);
30        int now=root;
31        for(int i=0;i<len;i++){
32            int c=s[i]-'a';
33            if(nx[now][c]==-1) return -1;
34            now=nx[now][c];
35        }
36        if(end[now]>0) return end[now];
37        else return -1;
38    }
39 }tr;

```

### 1.3 哈希

- 1 求hash值
- 2 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
- 3 ,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）
- 4 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：
- 5  $val["abc"] = 'a' * base^2 + 'b' * base^1 + 'c' * base^0$
- 6 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[l-1]部分）被多乘了许多次base，减的时候应该给hash[l-1]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，直到到hash[r]时已经

```

13 乘了(r-l+1)个base了, 实际上hash[r]=hash[l,r]+hash[l-1]*base^(r-l+1)所以答案应该是(多乘了的次
    数[l,r]区间长度)
14 val[i,r]=hash[r]-hash[l-1]*base^(r-l+1)最后, 因为乘的base一般很大, 所以乘多了容易爆, 要取模,
    为了避免麻烦, 一般使用
15 unsigned long long
16 Qhash如何支持单点修改?
17 A可以用线段树维护
18 要用线段树维护要资瓷区间合并>
19 hash=左子树hash*(base^右子树size)+右子树hash
20
21 struct HASH{
22     ull hash1;
23     ull p[N],ha[N];
24     void init(int n){
25         hash1=201326611;//233,50331653
26         p[0]=1; ha[0]=0;
27         for(int i=1;i<=n;i++)
28             p[i]=p[i-1]*hash1;
29     }
30     //传指针要从s开始, 不用s+1
31     void build(char *s,int x,int n){
32         for(int i=x;i<=n;i++) ha[i]=ha[i-1]*hash1+s[i];
33     }
34     ull getha(int l,int r){
35         return ha[r]-ha[l-1]*p[r-l+1];
36     }
37     int query(int x,int y){
38         int right=n-max(x,y)+1,left=1;
39         while(left<=right){
40             int mid=(left+right)>>1;
41             if(getha(x,x+mid-1)==getha(y,y+mid-1)) left=mid+1;
42             else right=mid-1;
43         }
44         return right;
45     }
46 }hs;
47
48 //将可以通过此类方法将二维、三维等压缩到一维ha数组中
49 int ha[10000007];
50 ll seed=309989,mod = 9989783;
51
52 inline int gethash(ll x,ll y){
53     int t=(x*seed+y)%mod;
54     return t;
55 }

```

## 1.4 AC 自动机

```

1 1、查询一个串可以匹配多少个串
2 int n;
3 char s[M];
4
5 //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
6 //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
7 //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
8 class tree {
9 public:
10     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);

```

```

70     int now = root;
71     for (int i = 0; i < len; i++) {
72         now = nx[now][s[i] - 'a'];
73         int tmp = now;
74         while (tmp != root && end[tmp] != 0) {
75             res += end[tmp];
76             end[tmp] = 0;
77             tmp = fail[tmp];
78         }
79     }
80     return res;
81 }
82 }ac;
83
84 int main() {
85     int t;
86     scanf("%d",&t);
87     while(t--){
88         ac.init();
89         scanf("%d",&n);
90         for(int i=1;i<=n;i++){
91             scanf("%s",s); ac.insert(s);
92         }
93         ac.build();
94         scanf("%s",s);
95         printf("%d\n",ac.query(s));
96     }
97 }
98
99 2、查询一个字符串中子串出现可重叠子串个数
100 int query(char s[]) {
101     int res = 0;
102     int len = strlen(s);
103     int now = root;
104     for (int i = 0; i < len; i++) {
105         now = nx[now][s[i]];
106         int tmp = now;
107         while (tmp != root) {
108             if(end[tmp]) v[end[tmp]]++;
109             tmp = fail[tmp];
110         }
111     }
112     return res;
113 }
114
115 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
116 通过ac自动机产生关系矩阵.
117 build函数中要加入: if (end[fail[now]]) end[now] = 1;
118 void pre_mat() {
119     memset(a.arr, 0, sizeof a.arr);
120     for (int i = 0; i < tot; i++) {
121         if (end[i]) continue;
122         for (int j = 0; j < 4; j++) {
123             int k = nx[i][j];
124             if (!end[k]) a.arr[i][k]++;
125         }
126     }
127 }

```

```

128
129 或当n小的时候使用dp即可实现
130 int n,m;
131 char s[N];
132
133 class tree {
134 public:
135     int nx[N][2], fail[N];
136     int end[N];
137     int root, tot;
138
139     int newNode() {
140         for (int i = 0; i < 2; i++)
141             nx[tot][i] = -1;
142         end[tot] = 0;
143         return tot++;
144     }
145
146     void init() {
147         tot = 0;
148         root = newNode();
149     }
150
151     void insert(char* s) {
152         int len = strlen(s);
153         int now = root;
154         for (int i = 0; i < len; i++) {
155             int id = s[i] - '0';
156             if (nx[now][id] == -1)
157                 nx[now][id] = newNode();
158             now = nx[now][id];
159         }
160         end[now] = 1;
161     }
162
163     void build() {
164         queue<int> q;
165         fail[root] = root;
166         for (int i = 0; i < 2; i++) {
167             if (nx[root][i] == -1) {
168                 nx[root][i] = root;
169             }
170             else {
171                 fail[nx[root][i]] = root;
172                 q.push(nx[root][i]);
173             }
174         }
175         while (!q.empty()) {
176             int now = q.front();
177             q.pop();
178             if (end[fail[now]]) end[now] = 1;
179             for (int i = 0; i < 2; i++) {
180                 if (nx[now][i] == -1) {
181                     nx[now][i] = nx[fail[now]][i];
182                 }
183                 else {
184                     fail[nx[now][i]] = nx[fail[now]][i];
185                     q.push(nx[now][i]);
186                 }
187             }
188         }
189     }
190 }

```

```

187     }
188 }
189 }
190 }ac;
191
192 ll dp[50][N];
193
194 int main() {
195     int t;
196     scanf("%d",&t);
197     while(t--){
198         scanf("%d%d%s",&n,&m,s);
199         ac.init();
200         ac.insert(s);
201         for(int i=0;i<n;i++){
202             if(s[i]=='1') s[i]='0';
203             else s[i]='1';
204             ac.insert(s);
205             if(s[i]=='0') s[i]='1';
206             else s[i]='0';
207         }
208         ac.build();
209         for (int i = 0; i <= m; i++) {
210             for (int j = 0; j < ac.tot; j++) {
211                 dp[i][j] = 0;
212             }
213         }
214         dp[0][0] = 1;
215         for(int i=1;i<=m;i++){
216             for(int j=0;j<ac.tot;j++){
217                 if(ac.end[j]) continue;
218                 for(int k=0;k<2;k++){
219                     int x=ac.nx[j][k];
220                     if(!ac.end[x]){
221                         dp[i][x]+=dp[i-1][j];
222                     }
223                 }
224             }
225         }
226         ll ret=0;
227         for(int i=0;i<ac.tot;i++) ret+=dp[m][i];
228         printf("%lld\n",pw(m)-ret);
229     }
230 }

```

232 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

233 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。

```

234 #include<bits/stdc++.h>

```

```

235 using namespace std;

```

```

236

```

```

237 typedef long long ll;

```

```

238 const int N = 1e5 + 10;

```

```

239 char str[N];

```

```

240 int n;

```

```

241 char s[2][N][8];

```

```

242 int p[2][N];

```

```

243 int cnt, cnt2;

```

```

244 int v[N],len,vis[N];

```

```

245 int l[N];
246 int vis2[N*10];
247
248
249 class tree {
250 public:
251     int nx[600005][26], fail[600005];
252     vector<int>end[600005];
253     int root, tot;
254
255     int newNode() {
256         for (int i = 0; i < 26; i++)
257             nx[tot][i] = -1;
258         end[tot].clear();
259         vis2[tot] = 0;
260         return tot++;
261     }
262
263     void init() {
264         tot = 0;
265         root = newNode();
266     }
267
268     void insert(char s[8], int x) {
269         int len = strlen(s);
270         int now = root;
271         for (int i = 0; i < len; i++) {
272             int id = s[i] - 'a';
273             if (nx[now][id] == -1)
274                 nx[now][id] = newNode();
275             now = nx[now][id];
276         }
277         end[now].push_back(x);
278     }
279
280     void build() {
281         queue<int>q;
282         fail[root] = root;
283         for (int i = 0; i < 26; i++) {
284             if (nx[root][i] == -1) {
285                 nx[root][i] = root;
286             }
287             else {
288                 fail[nx[root][i]] = root;
289                 q.push(nx[root][i]);
290             }
291         }
292         while (!q.empty()) {
293             int now = q.front();
294             q.pop();
295             for (int i = 0; i < 26; i++) {
296                 if (nx[now][i] == -1) {
297                     nx[now][i] = nx[fail[now]][i];
298                 }
299                 else {
300                     fail[nx[now][i]] = nx[fail[now]][i];
301                     q.push(nx[now][i]);
302                 }
303             }

```



```

304     }
305 }
306
307 void query1() {
308     int now = root;
309     for (int i = 0; i < len; i++) {
310         int id = str[i] - 'a';
311         now = nx[now][id];
312         int tmp = now;
313         while (tmp != root) {
314             if (end[tmp].size()) {
315                 vis2[tmp]++;
316             }
317             tmp = fail[tmp];
318         }
319     }
320     for (int i = 0; i < tot; i++) {
321         if (vis2[i]) {
322             for (int j = 0; j < end[i].size(); j++) {
323                 v[end[i][j]] += vis2[i];
324             }
325         }
326     }
327 }
328 void query2() {
329     int now = root; int step = 0;
330     for (int i = 0; i < len; i++) {
331         int id = str[i] - 'a'; step++;
332         now = nx[now][id];
333         int tmp = now;
334         while (tmp != root) {
335             if (end[tmp].size() && (step - vis[tmp]) >= l[end[tmp][0]]) {
336                 vis[tmp] = step;
337                 vis2[tmp]++;
338             }
339             tmp = fail[tmp];
340         }
341     }
342     for (int i = 0; i < tot; i++) {
343         if (vis2[i]) {
344             for (int j = 0; j < end[i].size(); j++) {
345                 v[end[i][j]] += vis2[i];
346             }
347         }
348     }
349 }
350 }ac;
351
352 int main() {
353     int t=1,pos;
354     while(scanf("%s",str)==1){
355         len = strlen(str);
356         cnt = cnt2 = 0;
357         scanf("%d", &n);
358         for (int i = 1; i <= n; i++) {
359             scanf("%d", &pos); v[i] = vis[i] = l[i] = 0;
360             if (pos == 0) {
361                 scanf("%s", s[pos][++cnt]);
362                 p[pos][cnt] = i;

```

```

363     }
364     else {
365         scanf("%s", s[pos][++cnt2]);
366         l[i] = strlen(s[pos][cnt2]);
367         p[pos][cnt2] = i;
368     }
369 }
370 ac.init();
371 for (int i = 1; i <= cnt; i++)
372     ac.insert(s[0][i], p[0][i]);
373 ac.build();
374 ac.query1();
375 ac.init();
376 for (int i = 1; i <= cnt2; i++)
377     ac.insert(s[1][i], p[1][i]);
378 ac.build();
379 ac.query2();
380 printf("Case %d\n", t++);
381 for (int i = 1; i <= n; i++)
382     printf("%d\n", v[i]);
383 printf("\n");
384 }
385 }
386
387 #include <stdio>
388 #include <cstring>
389 #include <queue>
390 #include <algorithm>
391 #define MAXN 600000+10
392 #define INF 0x3f3f3f3f
393 using namespace std;
394 int ans[MAXN][2];
395 int node[100000+10]; //记录串在Trie中的结束点
396 int n;
397 int op[100000+10];
398 struct Trie
399 {
400     int next[MAXN][26], fail[MAXN];
401     int pos[MAXN]; //记录当前节点的字符在模式串的位置
402     int last[MAXN]; //记录当前节点上一个匹配的位置
403     int L, root;
404     int newnode()
405     {
406         for(int i = 0; i < 26; i++)
407             next[L][i] = -1;
408         //End[L++] = 0;
409         pos[L++] = 0; //这里忘写了, MLE到死。。。
410         return L-1;
411     }
412     void init()
413     {
414         L = 0;
415         root = newnode();
416     }
417     void Insert(char *s, int id)
418     {
419         int now = root;
420         for(int i = 0; s[i]; i++)
421             {

```

```

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 }

```

```

481 };
482 Trie ac;
483 char str[100000+10];
484 char s[10];
485 int main()
486 {
487     int k = 1;
488     while(scanf("%s", str) != EOF)
489     {
490         ac.init(); scanf("%d", &n);
491         for(int i = 0; i < n; i++)
492         {
493             scanf("%d%s", &op[i], s);
494             ac.Insert(s, i);
495         }
496         ac.Build(); ac.solve(str);
497         printf("Case %d\n", k++);
498         for(int i = 0; i < n; i++)
499             printf("%d\n", ans[node[i]][op[i]]);
500         printf("\n");
501     }
502     return 0;
503 }

```

## 1.5 回文自动机

1 裸模板

2 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）

3 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）。

4 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）。

5 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）

6 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。

7 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。

8 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））。

9 8.p表示添加的节点个数。

10 9.tot表示添加的字符个数。

11 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

```

12 class PalindromicTree{
13 public:
14     int nx[N][dif], fail[N], len[N], cnt[N], num[N];
15     int tot, p, last, s[N];
16     int newnode(int l){
17         memset(nx[p], 0, sizeof(nx[p]));
18         len[p]=l;
19         cnt[p]=num[p]=0;
20         return p++;
21     }
22     void init(){
23         tot=p=last=0;
24         s[0]=-1, fail[0]=1;
25         newnode(0);
26         newnode(-1);
27     }
28     int getfail(int x){
29         while(s[tot-len[x]-1] != s[tot])
30             x=fail[x];
31         return x;

```

```

32     }
33     void insert(int x){
34         s[++tot]=x;
35         int cur = getfail(last);
36         int now = nx[cur][x];
37         if(!now){
38             now = newnode(len[cur]+2);
39             fail[now]=nx[getfail(fail[cur])][x];
40             nx[cur][x]=now;
41             num[now]=num[fail[now]]+1;
42         }
43         last=nx[cur][x];
44         cnt[last]++;
45     }
46     void makecnt(){
47         for(int i=p-1;i>=2;i--)
48             cnt[fail[i]]+=cnt[i];
49     }
50 }pt;
51
52 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
53 const int dif = 26;
54 int n, q;
55 int op;
56 char str[5];
57
58
59 class PalindromicTree{
60 public:
61     int nx[N][dif], fail[N], len[N], num[N];
62     int tot[2], p, last[2], s[N];
63     int newnode(int l){
64         memset(nx[p], 0, sizeof(nx[p]));
65         len[p]=l;
66         num[p]=0;
67         return p++;
68     }
69     void init(int x){
70         memset(s, -1, sizeof(s));
71         last[0]=last[1]=p=0;
72         tot[0]=x; tot[1]=x-1;
73         fail[0]=fail[1]=1;
74         newnode(0);
75         newnode(-1);
76     }
77     int getfail(int x, int tag){
78         if(!tag){
79             while(s[tot[tag]+len[x]+1]!=s[tot[tag]])
80                 x=fail[x];
81         }else{
82             while(s[tot[tag]-len[x]-1]!=s[tot[tag]])
83                 x=fail[x];
84         }
85         return x;
86     }
87     int insert(int x, int tag){
88         if(!tag)
89             s[--tot[0]]=x;
90         else

```

```

91         s[++tot[1]]=x;
92     int cur = getfail(last[tag],tag);
93     int now = nx[cur][x];
94     if(!now){
95         now = newnode(len[cur]+2);
96         fail[now]=nx[getfail(fail[cur],tag)][x];
97         nx[cur][x]=now;
98         num[now]+=num[fail[now]]+1;
99     }
100     last[tag]=nx[cur][x];
101     if(len[last[tag]]==tot[1]-tot[0]+1)
102         last[tag^1]=last[tag];
103     return num[last[tag]];
104 }
105 }pt;
106
107
108 int main() {
109     while(scanf("%d",&q)==1){
110         pt.init(q);
111         ll ans=0;
112         while(q--){
113             scanf("%d",&op);
114             if(op==1){
115                 scanf("%s",str);
116                 ans+=pt.insert(str[0]-'a',0);
117             }else if(op==2){
118                 scanf("%s",str);
119                 ans+=pt.insert(str[0]-'a',1);
120             }else if(op==3){
121                 printf("%d\n",pt.p-2);
122             }else{
123                 printf("%lld\n",ans);
124             }
125         }
126     }
127     return 0;
128 }
129
130 3、使用vector优化内存
131 class PalindromicTree{
132 public:
133     vector<pii>nx[N];
134     int fail[N],len[N],num[N];
135     int tot,p,last,s[N];
136     int newnode(int l){
137         nx[p].clear();
138         len[p]=l;
139         num[p]=0;
140         return p++;
141     }
142     void init(){
143         tot=p=last=0;
144         s[0]=-1,fail[0]=1;
145         newnode(0);
146         newnode(-1);
147     }
148     int getfail(int x){
149         while(s[tot-len[x]-1]!=s[tot])

```

```

150         x=fail[x];
151     return x;
152 }
153
154 int is_exist(int p,int c){
155     for(auto t:nx[p]){
156         if(t.first==c)
157             return t.second;
158     }
159     return 0;
160 }
161
162 void insert(int x){
163     s[++tot]=x;
164     int cur = getfail(last);
165     int now = is_exist(cur,x);
166     if(!now){
167         now = newnode(len[cur]+2);
168         fail[now]=is_exist(getfail(fail[cur]),x);
169         nx[cur].push_back(make_pair(x,now));
170         num[now]=num[fail[now]]+1;
171     }
172     last=now;
173 }
174 }pt;

```

## 1.6 后缀数组、后缀自动机

```

1  后缀数组模板
2  int n,m;
3  int sa[N],c[N],wa[N],wb[N];
4  int rk[N],height[N];
5  //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
6  //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
7  //height[i]为LCP(i,i-1), 1<i<=n, 显然height[1]=0;
8  //设h[i]=height[rk[i]], 同样的, height[i]=h[sa[i]]; 则有h[i]>=h[i-1]-1;
9
10 void DA(int *s,int n,int m){
11     //如果多组数据,x需要初始化
12     for(int i=0;i<=m;i++) c[i]=wa[i]=0;
13     int *x=wa,*y=wb;
14     for(int i=1;i<=n;i++) ++c[x[i]=s[i]];
15     for(int i=2;i<=m;i++) c[i]+=c[i-1];
16     for(int i=n;i>=1;i--) sa[c[x[i]]--]=i;
17     for(int k=1;k<=n;k<=1){
18         int p = 0;
19         for(int i=n-k+1;i<=n;i++) y[++p]=i;
20         for(int i=1;i<=n;i++) if(sa[i]>k) y[++p] = sa[i]-k;
21         for(int i=0;i<=m;i++) c[i]=0;
22         for(int i=1;i<=n;i++) ++c[x[i]];
23         for(int i=2;i<=m;i++) c[i]+=c[i-1];
24         for(int i=n;i>=1;i--) sa[c[x[y[i]]]--]=y[i];
25         swap(x,y);
26         x[sa[1]]=1;
27         p=1;
28         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;
29         if(p==n) break;

```

```

30     m=p;
31 }
32 int k=0;
33 for (int i=1; i<=n; ++i) rk[sa[i]]=i;
34 for (int i=1; i<=n; ++i) {
35     if (rk[i]==1) continue; //第一名height为0
36     if (k) --k; //h[i]>=h[i-1]-1;
37     int j=sa[rk[i]-1];
38     while (j+k<=n && i+k<=n && s[i+k]==s[j+k]) ++k;
39     height[rk[i]]=k; //h[i]=height[rk[i]];
40 }
41 }
42
43 int main() {
44     scanf("%s",s+1);
45     int n = strlen(s+1);
46     //注意下标从s开始, 而不是s+1
47     DA(s,n,127);
48     for(int i=1;i<=n;i++) printf("%d ",sa[i]);
49 }
50
51 后缀数组dc3:
52 #include <cstdio>
53 #include <cstring>
54 #include <algorithm>
55 #define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
56 #define G(x) ((x) < tb ? (x) * 3 + 1 : ((x) - tb) * 3 + 2)
57 using namespace std;
58 //开三倍空间
59 const int N = 3000005;
60 int wa[N], wb[N], wss[N], wv[N], sa[N];
61 int rnk[N], height[N], s[N];
62 char str[N];
63 //sa和rnk实际上是从[0,len]的所有值, 而s[len]=0,height正常
64
65
66 int c0(int *r, int a, int b) {
67     return r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2];
68 }
69
70 int c12(int k, int *r, int a, int b) {
71     if (k == 2)
72         return r[a] < r[b] || r[a] == r[b] && c12(1, r, a + 1, b + 1);
73     return r[a] < r[b] || r[a] == r[b] && wv[a + 1] < wv[b + 1];
74 }
75
76 void Rsort(int *r, int *a, int *b, int n, int m) {
77     for (int i = 0; i < n; i++) wv[i] = r[a[i]];
78     for (int i = 0; i < m; i++) wss[i] = 0;
79     for (int i = 0; i < n; i++) wss[wv[i]]++;
80     for (int i = 1; i < m; i++) wss[i] += wss[i - 1];
81     for (int i = n - 1; i >= 0; i--) b[--wss[wv[i]]] = a[i];
82 }
83
84 void dc3(int *r, int *sa, int n, int m) {
85     int i, j, *rn = r + n, *san = sa + n, ta = 0, tb = (n + 1) / 3, tbc = 0, p;
86     r[n] = r[n + 1] = 0;
87     for (i = 0; i < n; i++) if (i % 3 != 0) wa[tbc++] = i;
88     Rsort(r + 2, wa, wb, tbc, m);

```



```

89     Rsort(r + 1, wb, wa, tbc, m);
90     Rsort(r, wa, wb, tbc, m);
91     for (p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++)
92         rn[F(wb[i])] = c0(r, wb[i - 1], wb[i]) ? p - 1 : p++;
93     if (p < tbc) dc3(rn, san, tbc, p);
94     else for (i = 0; i < tbc; i++) san[rn[i]] = i;
95     for (i = 0; i < tbc; i++) if (san[i] < tb) wb[ta++] = san[i] * 3;
96     if (n % 3 == 1) wb[ta++] = n - 1;
97     Rsort(r, wb, wa, ta, m);
98     for (i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;
99     for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++)
100         sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i++] : wb[j++];
101     for (; i < ta; p++) sa[p] = wa[i++];
102     for (; j < tbc; p++) sa[p] = wb[j++];
103 }
104
105 void calheight(int *r, int *sa, int n) {
106     int i, j, k = 0;
107     for (i = 1; i <= n; i++) rnk[sa[i]] = i;
108     for (i = 0; i < n; height[rnk[i++]] = k)
109         for (k ? k-- : 0, j = sa[rnk[i] - 1]; r[i + k] == r[j + k]; k++);
110 }
111
112 int main() {
113     while (scanf("%s", str) == 1 && str[0] != '.') {
114         int len = strlen(str);
115         for (int i = 0; i < len; i++)
116             s[i] = str[i] - 'a' + 1;
117         s[len] = 0;
118         dc3(s, sa, len + 1, 105);
119         calheight(s, sa, len);
120         int aa = len - height[rnk[0]];
121         int ans = 1;
122         if (len % aa == 0) {
123             ans = len / aa;
124         }
125         printf("%d\n", ans);
126     }
127     return 0;
128 }
129
130 后缀自动机:
131 //凡是和后缀自动机相关的数组都必须开2倍
132 int n;
133 char s[N];
134 int sz[N<<1], c[N<<1], rk[N<<1];
135
136 class SuffixAutoMaton{
137 public:
138     int last, tot;
139     int nx[N<<1][dif], fa[N<<1], len[N<<1];
140     void init(){
141         last=tot=1;
142         fa[1]=len[1]=0;
143         memset(nx[1], 0, sizeof(nx[1]));
144     }
145     inline void insert(int c){
146         int p=last, np=++tot;
147         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;

```

```

207         else{
208             int nq=++tot;
209             len[nq]=len[p]+1;
210             nx[nq]=nx[q];
211             fa[nq]=fa[q]; fa[q]=fa[np]=nq;
212             for(;;p=fa[p]) {
213                 auto x=nx[p].lower_bound(make_pair(c,0));
214                 if((*x).second!=q){
215                     break;
216                 }else{
217                     nx[p].erase(x);
218                     nx[p].insert(make_pair(c,nq));
219                 }
220             }
221         }
222     }
223 }
224 }SAM;
225
226 //set优化vector
227 vector<pii>::iterator ite;
228
229 class SuffixAutoMaton{
230 public:
231     int last,tot;
232     vector<pii>nx[N<<1];
233     int fa[N<<1],len[N<<1];
234     ll ret;
235     void init(){
236         last=tot=1;
237         fa[1]=len[1]=0; nx[1].clear(); ret=0;
238     }
239
240     int isOK(int p,int c){
241         for(auto x:nx[p]){
242             if(x.first==c) return x.second;
243         }
244         return 0;
245     }
246
247     void Del(int p,int c){
248         for(ite=nx[p].begin();ite!=nx[p].end();ite++){
249             if((*ite).first==c){
250                 nx[p].erase(ite);return;
251             }
252         }
253     }
254
255     inline void insert(int c){
256         int p=last,np=++tot;
257         nx[np].clear();
258         last=np; len[np]=len[p]+1;
259         for(;!isOK(p,c);p=fa[p]) nx[p].push_back(make_pair(c,np));
260         if(!p) fa[np]=1;
261         else{
262             int q=isOK(p,c);
263             if(len[p]+1==len[q]) fa[np]=q;
264             else{
265                 int nq=++tot;

```

```

266         len[nq]=len[p]+1;
267         nx[nq]=nx[q];
268         fa[nq]=fa[q]; fa[q]=fa[np]=nq;
269         for(;isOK(p,c)==q;p=fa[p]) {
270             Del(p,c);
271             nx[p].push_back(make_pair(c,nq));
272         }
273     }
274 }
275 ret+=len[np]-len[fa[np]];
276 }
277 }SAM;
278
279 //广义后缀自动机
280
281 1、直接建立新节点
282 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。
283 //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
284 //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。
285 class SuffixAutoMaton{
286 public:
287     int last,tot;
288     int nx[N<<1][dif],fa[N<<1],len[N<<1];
289     ll sz[2][N<<1];
290     void init(){
291         last=tot=1;
292         fa[1]=len[1]=0;
293         memset(nx[1],0, sizeof(nx[1]));
294     }
295     inline void insert(int c,int op){
296         int p=last,np=++tot;
297         memset(nx[np],0,sizeof(nx[np]));
298         last=np; len[np]=len[p]+1;
299         for(;p&&!nx[p][c];p=fa[p]) nx[p][c]=np;
300         if(!p)
301             fa[np]=1;
302         else{
303             int q=nx[p][c];
304             if(len[p]+1==len[q]) fa[np]=q;
305             else{
306                 int nq=++tot;
307                 len[nq]=len[p]+1;
308                 memcpy(nx[nq],nx[q], sizeof(nx[q]));
309                 fa[nq]=fa[q]; fa[q]=fa[np]=nq;
310                 for(;nx[p][c]==q;p=fa[p])
311                     nx[p][c]=nq;
312             }
313         }
314         sz[op][np]=1;
315     }
316
317     int head[N<<1],cnt;
318     struct Edge{
319         int to,nx;
320     }e[N];
321

```

```

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     }
356     SAM.last=p=1;
357     scanf("%s",s);
358     n=strlen(s);
359     for(int i=0;i<n;i++){
360         SAM.insert(s[i]-'a'),p=SAM.nx[p][s[i]-'a'],SAM.sz[1][p]++;
361     }
362     SAM.calc();
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             }

```

```

381     int np=++tot;
382     memset(nx[tot],0, sizeof(nx[tot]));
383     last=np; len[np]=len[p]+1;
384     for(;p&&!nx[p][c];p=fa[p]) nx[p][c]=np;
385     if(!p) fa[np]=1;
386     else{
387         int q=nx[p][c];
388         if(len[p]+1==len[q]) fa[np]=q;
389         else{
390             int nq=++tot;
391             len[nq]=len[p]+1;
392             memcpy(nx[nq],nx[q], sizeof(nx[q]));
393             fa[nq]=fa[q]; fa[q]=fa[np]=nq;
394             for(;nx[p][c]==q;p=fa[p])
395                 nx[p][c]=nq;
396         }
397     }
398 }
399
400 void query(){
401     ll ret=0;
402     for(int i=1;i<=tot;i++) c[len[i]]++;
403     for(int i=1;i<=tot;i++) c[i]+=c[i-1];
404     for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
405     for(int i=tot;i>=1;i--){
406         int p=rk[i];
407         sz[0][fa[p]]+=sz[0][p]; sz[1][fa[p]]+=sz[1][p];
408         ret+=(ll)sz[0][p]*sz[1][p]*(len[p]-len[fa[p]]);
409     }
410     printf("%lld\n",ret);
411 }
412
413 }SAM;
414
415 int main(){
416     SAM.init();
417     scanf("%s",s);
418     int len=strlen(s),p=1;
419     for(int i=0;i<len;i++){
420         int c=s[i]-'a';SAM.insert(c);
421         p=SAM.nx[p][c];SAM.sz[0][p]++;
422     }
423     SAM.last=p=1;
424     scanf("%s",s);
425     len=strlen(s);
426     for(int i=0;i<len;i++){
427         int c=s[i]-'a';SAM.insert(c);
428         p=SAM.nx[p][c];SAM.sz[1][p]++;
429     }
430     SAM.query();
431 }
432 }

```

## 2 动态规划

### 2.1 背包 dp

```

1 //m-背包容量, w-物品体积, val-物品价值, cnt-物品数量
2 //01背包
3 void OneZero(int m,int w,int val){
4     for(int i=m;i>=w;i--)
5         dp[i]=max(dp[i-w]+val,dp[i]);
6 }
7
8 //完全背包
9 void Com(int m,int w,int val){
10     for(int i=0;i<=m-w;i++)
11         dp[i+w]=max(dp[i]+val,dp[i+w]);
12 }
13
14 //多重背包
15 void Mul(int m,int w,int val,int cnt){
16     if(cnt*w>=m){
17         Com(m,w,val);
18         return;
19     }
20     for(int i=1;i<=cnt;i<=1){
21         OneZero(m,w*i,val*i); cnt-=i;
22     }
23     if(cnt) OneZero(m,cnt*w,val*cnt);
24 }

```

### 2.2 数位 dp

```

1 int gcd(int a, int b) {
2     return b ? gcd(b, a%b) : a;
3 }
4
5 ll dfs(int pos, int tot, int lcm, bool limit) {
6     if (pos == 0)
7         return (tot%lcm == 0);
8     if (!limit&&dp[pos][ha[lcm]][tot] != -1)
9         return dp[pos][ha[lcm]][tot];
10    ll res = 0;
11    int top = limit ? di[pos] : 9;
12    for (int i = 0; i <= top; i++) {
13        res += dfs(pos - 1, (tot * 10 + i) % mod, i ? i * lcm / gcd(i, lcm) : lcm, i ==
14            di[pos] && limit);
15    }
16    if (!limit)
17        dp[pos][ha[lcm]][tot] = res;
18    return res;
19 }
20 ll solve(ll x) {
21     int pos = 0;
22     while (x) {
23         di[++pos] = x % 10;
24         x /= 10;
25     }
26     return dfs(pos, 0, 1, true);

```

```
27 }
```

## 2.3 dp 优化

```

1 1、斜率优化
2 int tail=0,head=0;
3 q[tail++]=0;
4 for(int i=1;i<=n;i++){
5     //判断答案是否最优
6     while(tail>1+head&&isOK(i,q[head+1],q[head])) head++;
7     dp[i]=getsum(i,q[head]);
8     //判断新加入的点是否最优
9     while(tail>1+head&&isOK2(i,q[tail-1],q[tail-2])) tail--;
10    q[tail++]=i;
11 }
12 printf("%lld\n",dp[n]);
13
14 2、四边形不等式
15 在dp问题中,我们常见这样的一类问题,他们的dp转移方程式这样的:  $dp[i][j]=\min\{dp[i][k]+dp[k+1][j]+cost[i][j]\}$ 
16
17 对于( $a<b\leq c<d$ ),如果有 $f[a][c]+f[b][d]\leq f[b][c]+f[a][d]$ ,则说明f满足四边形不等式
18 1、当决策代价函数 $w[i][j]$ 满足 $w[i][j]+w[i'][j']\leq w[i'][j]+w[i][j']$  ( $i\leq i'\leq j\leq j'$ )时,称满足四边形不等式
19 2、当函数 $w[i][j]$ 满足 $w[i'][j]\leq w[i][j']$  ( $i\leq i'\leq j\leq j'$ )时,称w关于区间包含关系单调.
20 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
21  $dp[i][j]$ 取得最优值时对应的下标,即( $i\leq k\leq j$ )。就满足 $s[i][j-1]\leq s[i][j]\leq s[i+1][j]$ 。
22
23 一般做法:
24 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
25 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。
26
27 //判断是否满足平行四边形不等式优化 $w[i][j]+w[i+1][j+1]\leq w[i+1][j]+w[i][j+1]$ 
28 bool isOK(){
29     for(int i=1;i<n;i++){
30         for(int j=i+2;j<n;j++){
31             if( $w[i][j]+w[i+1][j+1]\leq w[i+1][j]+w[i][j+1]$ ) continue;
32             else return false;
33         }
34     }
35     return true;
36 }
37
38
39 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
40 知道,满足这些性质的转移方程不止这一种! 对于 $f[i][j] = \min\{ f[i-1][k] + w(k+1,j) \mid i-1 \leq k < j \}$ 这个方程来说,若w满足四
41 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!
42 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
43 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

```



45 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  
 46 对于区间dp，行号大于列号  
 47 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。  
 47 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  
 48 调递增，所以应该把k的枚举  
 48 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  
 49 范围。而且，如果我们遵循这  
 49 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

1 编辑距离：  
 2 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  
 3 少的编辑操  
 3 作个数称为S和S2的编辑距离。  
 4  
 5 `int main(){`  
 6     `scanf("%s%s",s+1,s2+1);`  
 7     `int len=strlen(s+1),len2=strlen(s2+1);`  
 8     `dp[1][1]=s[1]==s2[1]?0:1;`  
 9     `for(int i=2;i<=len;i++) dp[i][1]=dp[i-1][1]+1;`  
 10     `for(int j=2;j<=len2;j++) dp[1][j]=dp[1][j-1]+1;`  
 11     `for(int i=2;i<=len;i++){`  
 12         `for(int j=2;j<=len2;j++){`  
 13             `dp[i][j]=min(dp[i-1][j],min(dp[i][j-1],dp[i-1][j-1]))+1;`  
 14             `if(s[i]==s2[j]) dp[i][j]=min(dp[i][j],dp[i-1][j-1]);`  
 15         `}`  
 16     `}`  
 17     `for(int i=1;i<=len;i++){`  
 18         `for(int j=1;j<=len2;j++){`  
 19             `printf("%d ",dp[i][j]);`  
 20         `}`  
 21         `printf("\n");`  
 22     `}`  
 23 `}`

## 2.5 状压 dp

1 遍历方式：  
 2 //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  
 3 //第二层可以检验某一位是否存在，从而做出操作  
 4 `for(int i=0;i<pw(n);i++){`  
 5     `for(int j=0;j<n;j++){`  
 6         `if(pw(j)&i){`  
 7             .....  
 8         `}`  
 9     `}`  
 10 `}`  
 11  
 12  
 13 //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  
 14 `int main(){`  
 15     `int x;`  
 16     `scanf("%d%d",&n,&m);`  
 17     `int top=pw(m);`  
 18     `for(int i=1;i<=n;i++){`  
 19         `for(int j=1;j<=m;j++){`

```

20         scanf("%d",&x);
21         g[i]=(g[i]<<1)+x; //预处理状态, 将其直接保存
22     }
23 }
24 a[0]=0;
25 for(int i=0;i<top;i++){
26     if((((i<<1)&i)==0)&&(((i>>1)&i)==0)){
27         a[++a[0]]=i; //通过移位操作, 预处理出相邻不为1的状态
28     }
29 }
30 dp[0][0]=1;
31 for(int i=1;i<=n;i++){
32     //遍历矩阵上下关系来处理
33     for(int j=1;j<=a[0];j++){
34         ...
35         for(int k=1;k<=a[0];k++){
36             ...
37         }
38     }
39 }
40 ll ret=0; //答案统计
41 for(int i=0;i<top;i++) ret=(ret+dp[n][i])%mod;
42 printf("%lld\n",ret);
43 }
44
45 //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
46 for(int i=0;i<n;i++) if(p[i]==-1||p[i]==0) dp[pw(i)][i]=0;
47 for(int i=0;i<top;i++){
48     for(int j=0;j<n;j++){
49         if(dp[i][j]==-inf) continue;
50         for(int k=0;k<n;k++){
51             if(k==j) continue;
52             if(p[k]==-1||p[k]==cnt[i]){
53                 if(pw(k)&i) continue;
54                 dp[i|pw(k)][k]=max(dp[i|pw(k)][k],dp[i][j]+a[j]*a[k]);
55             }
56         }
57     }
58 }
59
60
61 //可以快速求出所有关于S的二进制子集
62 int S=i;
63 for (int s=(S-1)&S;s;s=(s-1)&S) {
64     int t=S^s;
65 }

```

### 3 数据结构

#### 3.1 并查集

```

1 1、并查集模板
2 int find(int x){
3     if (x != parent[x])
4         parent[x] = find(parent[x]);
5     return parent[x];
6 }
7
8 2、带权并查集
9 int find(int x){
10     if (x != parent[x]){
11         int t = parent[x];
12         parent[x] = find(parent[x]);
13         value[x] += value[t];
14     }
15     return parent[x];
16 }
17
18 void Merge(int x,int y,int v){
19     int fx = find(x);
20     int fy = find(y);
21     if (fx != fy){
22         parent[fx] = fy;
23         value[fx] = -value[x] + value[y] + v;
24     }
25 }

```

#### 3.2 线段树

```

1 struct SegTree{
2     ll sum[N<<2];
3
4     static inline int lson(int k) {return k<<1;}
5
6     static inline int rson(int k) {return k<<1|1;}
7
8     void up(int k){
9         sum[k]=sum[lson(k)]+sum[rson(k)];
10    }
11
12    void build(int k,int l,int r){
13        if(l==r){
14            scanf("%lld",&sum[k]);
15            return;
16        }
17        int mid=(l+r)>>1;
18        build(lson(k),l,mid);
19        build(rson(k),mid+1,r);
20        up(k);
21    }
22
23    void update(int k,int l,int r,int pos,int val){
24        if(l==r){
25            sum[k]=max(0ll,sum[k]+val);
26            return;

```

```

27     }
28     int mid=(l+r)>>1;
29     if(pos<=mid) update(lson(k),l,mid,pos,val);
30     else update(rson(k),mid+1,r,pos,val);
31     up(k);
32 }
33
34 ll query(int k,int l,int r,int x,int y){
35     if(l==x&&r==y){
36         return sum[k];
37     }
38     int mid = (l+r)>>1;
39     if(y<=mid) return query(lson(k),l,mid,x,y);
40     else if(x>mid) return query(rson(k),mid+1,r,x,y);
41     else return query(lson(k),l,mid,x,mid)+query(rson(k),mid+1,r,mid+1,y);
42 }
43 }st;

```

### 3.3 st 表

```

1 1、st表
2 struct ST {
3     int k2[21], st[21][N], Log[N];
4     void init_st(int n) {
5         k2[0] = 1;
6         for (int i = 1; i <= 20; i++) k2[i] = 2 * k2[i - 1];
7         Log[0] = -1; for (int i = 1; i < N; i++) Log[i] = Log[i / 2] + 1;
8         for (int i = 1; i <= n; i++) st[0][i] = height[i];
9         for (int i = 1; i <= Log[n]; i++) {
10             for (int j = 1; j + k2[i] - 1 <= n; j++) {
11                 st[i][j] = min(st[i - 1][j], st[i - 1][j + k2[i - 1]]);
12             }
13         }
14     }
15     int query_min(int x, int y) {
16         int len = log2(y - x + 1);
17         return min(st[len][x], st[len][y - k2[len] + 1]);
18     }
19 }st;
20
21 精简版
22 scanf("%d%d",&n,&m);
23 for(int i=1;i<=n;i++)
24     scanf("%d",&st[0][i]);
25 int top = log2(n);
26 for(int i=1;i<=top;i++){
27     for(int j=1;j+(1<<i)-1<=n;j++){
28         st[i][j]=min(st[i-1][j],st[i-1][j+(1<<i-1)]);
29     }
30 }
31 int x=1,y=m;
32 while(y<=n){
33     int len = log2(y-x+1);
34     printf("%d\n",min(st[len][x],st[len][y-(1<<len)+1]));
35     x++;y++;
36 }

```

### 3.4 线性基

```

1 struct Linear_Basis{
2     ll d[61],p[61];
3     int cnt;
4     bool zero;
5     //初始化
6     Linear_Basis(){
7         memset(d,0,sizeof(d));
8         memset(p,0,sizeof(p));
9         cnt=0; zero=0;
10    }
11    //插入
12    bool insert(ll val){
13        for (int i=60;i>=0;i--){
14            if (val&(1LL<<i)){
15                if (!d[i]){
16                    d[i]=val;
17                    break;
18                }
19                val^=d[i];
20            }
21            return val>0;
22        }
23    //查询线性基所能表示最大值
24    ll query_max(){
25        ll ret=0;
26        for (int i=60;i>=0;i--){
27            if ((ret^d[i])>ret)
28                ret^=d[i];
29        }
30        return ret;
31    //查询线性基所能表示最小值
32    ll query_min(){
33        for (int i=0;i<=60;i++){
34            if (d[i])
35                return d[i];
36        }
37        return 0;
38    //重构,消除多余的1
39    void rebuild(){
40        for (int i=60;i>=0;i--){
41            for (int j=i-1;j>=0;j--){
42                if (d[i]&(1LL<<j))
43                    d[i]^=d[j];
44            }
45            for (int i=0;i<=60;i++){
46                if (d[i])
47                    p[cnt++]=d[i];
48            }
49    //求第k小数,注意特判0
50    ll kthquery(ll k){
51        ll ret=0;
52        if (k>=(1LL<<cnt))
53            return -1;
54        for (int i=60;i>=0;i--){
55            if (k&(1LL<<i))
56                ret^=p[i];
57        }
58    }

```

```

58 //查询是否存在x->y的变化
59 bool query_exist(ll x,ll y){
60     for(int i=60;i>=0;i--){
61         if((1ll<<i)&y){
62             if((1ll<<i)&x)
63                 continue;
64             else
65                 x^=d[i];
66         }else if((1ll<<i)&x){
67             x^=d[i];
68         }
69     }
70     return x==y;
71 }
72 };
73
74 //线性基合并
75 Linear_Basis merge(const Linear_Basis &n1,const Linear_Basis &n2){
76     Linear_Basis ret=n1;
77     for (int i=60;i>=0;i--){
78         if (n2.d[i])
79             ret.insert(n1.d[i]);
80     }
81     return ret;
82 }

```

### 3.5 树分治

```

1 //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。
2 int n,k,head[N],tot;
3 bool vis[N];
4 int part[N],cnt[N],root,deep[N],d[N],dtot,sn;
5 ll ans;
6 struct node{
7     int to,nx,val;
8 }edge[N*2];
9
10 void add_edge(int from,int to,int val){
11     edge[tot].to=to;
12     edge[tot].val=val;
13     edge[tot].nx=head[from];
14     head[from]=tot++;
15 }
16
17 //求取重心
18 void getRoot(int from,int pre){
19     cnt[from]=1; part[from]=0;
20     for(int i=head[from];i;i=edge[i].nx){
21         int to=edge[i].to;
22         if(to==pre||vis[to]) continue;
23         getRoot(to,from); cnt[from]+=cnt[to];part[from]=max(part[from],cnt[to]);
24     }
25     part[from]=max(part[from],sn-cnt[from]);
26     if(part[root]>part[from]) root=from;
27 }
28
29 //求取树深度
30 void getDeep(int from,int pre){

```

```

31     d[++dtot]=deep[from];
32     for(int i=head[from];i;i=edge[i].nx){
33         int to=edge[i].to;
34         if(to==pre||vis[to]) continue;
35         deep[to]=deep[from]+edge[i].val;
36         getDeep(to,from);
37     }
38 }
39
40 //计算答案贡献
41 ll calc(int from){
42     dtot=0; getDeep(from,0);
43     sort(d+1,d+1+dtot);
44     ll sum=0;
45     int i=1,j=dtot;
46     while(i<j){
47         if(d[i]+d[j]<=k) sum+=j-i,i++;
48         else j--;
49     }
50     return sum;
51 }
52
53 //分治
54 void Divide(int from){
55     deep[from]=0; ans+=calc(from); vis[from]=1;
56     for(int i=head[from];i;i=edge[i].nx){
57         int to=edge[i].to;
58         if(vis[to]) continue;
59         deep[to]=edge[i].val; ans-=calc(to); sn=part[to];
60         root=0; getRoot(to,0); Divide(root);
61     }
62 }
63
64 int main() {
65     part[0]=inf;
66     int u,v,w;
67     while(scanf("%d%d",&n,&k)==2){
68         if(n==0&&k==0) break;
69         tot=1;
70         memset(head,0, sizeof(int)*(n+1));
71         memset(vis,0, sizeof(bool)*(n+1));
72         for(int i=1;i<n;i++){
73             scanf("%d%d%d",&u,&v,&w);
74             add_edge(u,v,w); add_edge(v,u,w);
75         }
76         ans=root=0; sn=n;
77         getRoot(1,0); Divide(root);
78         printf("%lld\n",ans);
79     }
80     return 0;
81 }

```

### 3.6 树重心

```

1 int n;
2 vector<int>e[N];
3 int min_p,min_part;
4 int node[N];

```

```

5 //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
6 //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。
7 //性质：
8 //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。
9 //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。
10 //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。
11 //4、一棵树最多有两个重心，且相邻。
12
13 //min_p,min_part是求得重心
14
15 void dfs(int from,int pre){
16     node[from]=1; int max_part=0;
17     for(int i=0;i<e[from].size();i++){
18         int to=e[from][i];
19         if(pre==to) continue;
20         dfs(to,from);
21         node[from]+=node[to];
22         max_part=max(max_part,node[to]);
23     }
24     max_part=max(max_part,n-node[from]);
25     if(max_part<min_part){
26         min_p=from; min_part=max_part;
27     }
28 }
29
30 int main() {
31     int t,u,v;
32     scanf("%d",&t);
33     while(t--){
34         scanf("%d",&n);
35         for(int i=1;i<=n;i++) e[i].clear();
36         for(int i=1;i<n;i++){
37             scanf("%d%d",&u,&v);
38             e[u].push_back(v); e[v].push_back(u);
39         }
40         min_p=0; min_part=inf;
41         dfs(1,0);
42         printf("%d %d\n",min_p,min_part);
43     }
44     return 0;
45 }

```

### 3.7 树状数组

```

1 树状数组：
2     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。
3     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
4  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
5
6  //注意0
7  //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
8  struct BIT{
9      ll c[N];
10

```



```

11 inline int lowbit(int x) {return x&-x;}
12
13 void add(int x,int val){
14     for(;x<=n;x+=lowbit(x)){
15         c[x]+=val;
16     }
17 }
18
19 ll sum(int x){
20     ll sum=0;
21     for(;x>0;x-=lowbit(x)){
22         sum+=c[x];
23     }
24     return sum;
25 }
26 };
27

```

28 树状数组维护前缀和的前缀和:

29 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。  
 30 设s1表示数组a的前缀和, s2表示s1的前缀和, 则有:

31  $s1[i] = \sum_{j=1}^i a[j]$

32  $s2[i] = \sum_{j=1}^i s1[j]$

33 经化简可得:  $s2[i] = (i+1)\sum_{j=1}^i a[j] - \sum_{j=1}^i j*a[j]$

34 因此我们只需要开两个数组维护  $(i+1)\sum_{j=1}^i a[j]$  和  $\sum_{j=1}^i j*a[j]$  即可。

35

36 更高阶情况查看纸质模板

### 3.8 树的直径

1 树的直径:

2 概念:

3 树上距离最远的点 (树的最长路)。

4 思想

5 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。

6 过程:

7 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

8 为u, 第一次BFS找到的终点v一定是树的直径的一个端点。

9 证明:

10 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)

11 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

12 从v进行BFS得到的一定是直径长度

## 4 图论

### 4.1 最短路

```

1 1、Dijkstra
2 //不能处理负权图, 复杂度(V+E)logV
3 int n,m,tot,head[N];
4 ll dis[N];
5 bool vis[N];
6 struct node{
7     int id;
8     ll d;
9     node(){}
10    node(int id,ll d):id(id),d(d){}
11    bool operator < (const node& x) const {
12        return d>x.d;
13    }
14 };
15
16 struct edge{
17     int to,nx;
18     ll w;
19 }e[N<<1];
20
21 //tot初始化为1
22 void add_edge(int from,int to,ll w){
23     e[tot].to=to; e[tot].w=w;
24     e[tot].nx=head[from]; head[from]=tot++;
25 }
26
27 void Dijkstra(int st){
28     memset(dis,0x3f, sizeof(ll)*(n+1));
29     memset(vis,0, sizeof(bool)*(n+1));
30     dis[st]=0;
31     priority_queue<node>q;
32     node y(st,0);
33     q.push(y);
34     while(!q.empty()){
35         node x=q.top(); q.pop();
36         if(vis[x.id]) continue;
37         vis[x.id]=1;
38         for(int i=head[x.id];i;i=e[i].nx){
39             int to=e[i].to,w=e[i].w;
40             if(vis[to]) continue;
41             if(w+dis[x.id]<dis[to]){
42                 dis[to]=w+dis[x.id];
43                 y.d=dis[to]; y.id=to;
44                 q.push(y);
45             }
46         }
47     }
48 }
49
50 2、Floyd
51 //可以求任意两个点的最短路, 以及输出字典序的路径
52 int n;
53 int link[N][N],path[N][N];
54 int w[N];
55

```

```

56 void Floyd(int n){
57     for(int k=1;k<=n;k++){
58         for(int i=1;i<=n;i++){
59             for(int j=1;j<=n;j++){
60                 if(link[i][k]+link[k][j]+w[k]<link[i][j]){
61                     link[i][j]=link[i][k]+link[k][j]+w[k];
62                     path[i][j]=path[i][k];
63                 }else if(link[i][k]+link[k][j]+w[k]==link[i][j]&&path[i][j]>path[i][k])
64                     {
65                         path[i][j]=path[i][k];
66                     }
67             }
68         }
69     }
70 }
71 int main() {
72     int x;
73     while(scanf("%d",&n)==1){
74         if(n==0) break;
75         for(int i=1;i<=n;i++) for(int j=1;j<=n;j++){
76             scanf("%d",&x);
77             if(x==-1) link[i][j]=inf;
78             else link[i][j]=x;
79             path[i][j]=j;
80         }
81         for(int i=1;i<=n;i++) scanf("%d",&w[i]);
82         Floyd(n);
83         int start,endd;
84         while(scanf("%d%d",&start,&endd)==2){
85             if(start==endd&&start==-1) break;
86             printf("From %d to %d :\nPath: ",start,endd);
87             int x=start;
88             printf("%d",x);
89             while(x!=endd){
90                 printf("-->%d",path[x][endd]);
91                 x=path[x][endd];
92             }
93             printf("\nTotal cost : %d\n\n",link[start][endd]);
94         }
95     }
96 }
97 }
98
99 题意：
100     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。
101     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，如果有多条路径符合，则输出字典序最小的路径。
102
103
104 //输入
105 5
106 0 3 22 -1 4
107 3 0 5 -1 -1
108 22 5 0 9 20
109 -1 -1 9 0 4
110 4 -1 20 4 0
111 5 17 8 3 1
112 1 3

```

```

113 3 5
114 2 4
115 -1 -1
116 0
117
118 //输出
119 From 1 to 3 :
120 Path: 1-->5-->4-->3
121 Total cost : 21
122
123 From 3 to 5 :
124 Path: 3-->4-->5
125 Total cost : 16
126
127 From 2 to 4 :
128 Path: 2-->1-->5-->4
129 Total cost : 17
130
131 3、spfa
132 //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
133 bool spfa(int st){
134     memset(dis,0x3f, sizeof(ll)*(n+1));
135     memset(vis,0, sizeof(bool)*(n+1));
136     memset(cnt,0, sizeof(int)*(n+1));
137     queue<int>q;
138     vis[st]=1;dis[st]=0;q.push(st);cnt[st]++;
139     while(!q.empty()){
140         int from=q.front();
141         q.pop(); vis[from]=0;
142         for(int i=head[from];i;i=edge[i].nx){
143             int to=edge[i].to;
144             if(dis[to]>dis[from]+edge[i].w){
145                 dis[to]=dis[from]+edge[i].w;
146                 if(!vis[to]){
147                     vis[to]=1;q.push(to); cnt[to]=cnt[from]+1;
148                     if(cnt[to]>n) return true;
149                 }
150             }
151         }
152     }
153     return false;
154 }
155
156 //Floyd判负环
157 bool Floyd(int n){
158     for(int k=1;k<=n;k++){
159         for(int i=1;i<=n;i++){
160             for(int j=1;j<=n;j++){
161                 int t=link[i][k]+link[k][j];
162                 if(link[i][j]>t)link[i][j]=t;
163             }
164             if(link[i][i]<0) return true;
165         }
166     }
167     return false;
168 }

```

## 4.2 最小生成树

1 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。

```

2
3 1、 Prim
4 //时间复杂度 $(V+E)\log V$ 
5
6 struct node{
7     int id,w;
8     bool operator <(const node&x) const {
9         return w>x.w;
10    }
11 };
12
13 ll Prim(){
14     ll res=0; node x; int cnt=0;
15     priority_queue<node>q; q.push(node{1,0});
16     while(!q.empty()){
17         node y=q.top(); q.pop();
18         if(vis[y.id]) continue;
19         res+=y.w; vis[y.id]=1; cnt++;
20         for(int i=head[y.id]; i; i=edge[i].nx){
21             int to=edge[i].to,w=edge[i].w;
22             if(vis[to]) continue;
23             q.push(node{to,w});
24         }
25     }
26     if(cnt==n) return res;
27     return -1;
28 }
29
30 2、 Kruscal
31 //时间复杂度 $E\log E$ 
32 //将权值从大到小排列可生成最大生成树
33
34 int n,m,tot,parent[N];
35 struct node{
36     int from,to,w;
37 }edge[N];
38
39 int find(int x){
40     if (x != parent[x]) parent[x] = find(parent[x]);
41     return parent[x];
42 }
43
44 bool cmp(node x,node y){
45     return x.w<y.w;
46 }
47
48 ll Kruscal(){
49     ll res=0;
50     for(int i=1;i<=n;i++) parent[i]=i;
51     sort(edge+1,edge+1+m,cmp); int cnt=0;
52     for(int i=1;i<=m;i++){
53         if(cnt==n-1) return res;
54         int fx=find(edge[i].from),fy=find(edge[i].to);
55         if(fx!=fy){
56             parent[fx]=fy;

```

```

57         res+=edge[i].w;
58         cnt++;
59     }
60 }
61 if(cnt!=n-1) return -1;
62 else return res;
63 }
64
65 //检验最小生成树唯一性可以使用Kruskal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
66 ll Kruskal(){
67     ll ret=0; int cnt=0; v[0]=0;
68     for(int i=1;i<=n;i++) parent[i]=i;
69     sort(edge+1,edge+1+m,cmp);
70     for(int i=1;i<=m;i++){
71         if(cnt==n-1) break;
72         int fx=find(edge[i].from),fy=find(edge[i].to);
73         if(fx!=fy){
74             parent[fx]=fy; cnt++; ret+=edge[i].w; v[++v[0]]=i;
75         }
76     }
77     for(int i=1;i<=v[0];i++){
78         ll sum=0; cnt=0;
79         for(int j=1;j<=n;j++) parent[j]=j;
80         for(int j=1;j<=m;j++){
81             if(j==v[i]) continue;
82             int fx=find(edge[j].from),fy=find(edge[j].to);
83             if(fx!=fy){
84                 parent[fx]=fy; cnt++; sum+=edge[j].w;
85             }
86             if(cnt==n-1&&ret==sum) return -1;
87             else if(cnt==n-1) break;
88         }
89     }
90     return ret;
91 }

```

### 4.3 强联通分量

```

1 1、tarjan
2 //初始化dfs=top=cnt=0,tot=1;
3 //如果多组数据head[N], dfn[N], low[N], Stack[N], color[N]都得初始化
4 //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
5 //时间复杂度为O(v+e)
6 //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
7 int n, m;
8 int head[N], dfn[N], low[N], Stack[N], color[N];
9 bool vis[N];
10 int dfs, tot, top, cnt;
11
12 struct node {
13     int from, to, nx;
14 } edge[M];
15
16 void add_edge(int from, int to) {
17     edge[tot].from = from;
18     edge[tot].to = to;
19     edge[tot].nx = head[from];
20     head[from] = tot++;

```

```

21 }
22
23 void Tarjan(int x) {
24     dfn[x] = ++dfs; low[x] = dfs;
25     vis[x] = 1; Stack[++top] = x;
26     for (int i = head[x]; i; i = edge[i].nx) {
27         int tmp = edge[i].to;
28         if (!dfn[tmp]) {
29             Tarjan(tmp); low[x] = min(low[tmp], low[x]);
30         } else if (vis[tmp]) low[x] = min(low[x], dfn[tmp]);
31     }
32     if (dfn[x] == low[x]) {
33         vis[x] = 0; color[x] = ++cnt;
34         while (Stack[top] != x) {
35             color[Stack[top]] = cnt;
36             vis[Stack[top--]] = false;
37         }
38         top--;
39     }
40 }
41
42 void gao(){
43     //多组注意初始化
44     for(int i=1;i<=n;i++){
45         if(color[i]==0) Tarjan(i);
46     }
47     for(int i=1;i<=n;i++) printf("%d ",color[i]);
48     //重新建图, 进行操作
49 }
50
51 2、kosaraju
52 //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
53 //时间复杂度为O(v+e)
54
55 int n,m,cnt;
56 int head[N],head2[N],tot,tot2;
57 int color[N];
58 bool vis[N];
59 struct node{
60     int to,nx;
61 }edge[N],edge2[N];
62 stack<int>st;
63
64 void dfs1(int from){
65     vis[from]=1;
66     for(int i=head[from];i;i=edge[i].nx){
67         int to=edge[i].to;
68         if(!vis[to]) dfs1(to);
69     }
70     st.push(from);
71 }
72
73 void dfs2(int from){
74     color[from]=cnt;
75     for(int i=head2[from];i;i=edge2[i].nx){
76         int to=edge2[i].to;
77         if(!color[to]) dfs2(to);
78     }
79 }

```

```

80
81 void Kosaraju(){
82     memset(color,0, sizeof(int)*(n+1));
83     memset(vis,0, sizeof(bool)*(n+1)); cnt=0;
84     for(int i=1;i<=n;i++)
85         if(!vis[i]) dfs1(i);
86     while(!st.empty()){
87         int x=st.top(); st.pop();
88         if(!color[x]){
89             cnt++;
90             dfs2(x);
91         }
92     }
93 }

```

#### 4.4 网络流

```

1 1、dinic
2 struct Edge {
3     int e, nxt;
4     ll v;
5
6     Edge() = default;
7
8     Edge(int a, ll b, int c = 0) : e(a), v(b), nxt(c) {}
9
10    bool operator<(const Edge& a) const {
11        return (a.v == v ? e < a.e : v < a.v);
12    }
13
14    bool operator>(const Edge& a) const {
15        return (a.v == v ? e > a.e : v > a.v);
16    }
17 };
18
19 struct Graph {
20     Edge eg[M];
21     int head[N];
22     int cnt;
23
24     void init(int n) {
25         memset(head, -1, sizeof(int) * ++n);
26         cnt = 0;
27     }
28
29     inline void addEdge(int x, int y, ll v) {
30         eg[cnt] = Edge(y, v, head[x]);
31         head[x] = cnt++;
32     }
33 } gh;
34
35
36 struct Dinic {
37     Graph gh;
38     // 点的范围[0, n)
39     int n;
40     // 弧优化
41     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))
95         )) {
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);

```

```

100     }
101     return ans;
102 }
103 } dinic;

```

## 4.5 最小树形图

```

1  朱刘算法( $O(VE)$ )
2  一、相关定义
3  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  $v_i$ ，它不是任何弧的终
   点，而  $V$  中的
4  其它顶点都恰好是唯一的一条弧的终点，则称  $G$  是以  $v_i$  为根的树形图。
5  最小树形图就是有向图  $G = (V, E)$  中以  $v_i$  为根的树形图中权值和最小的那一个。
6  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总权值最
   小。
7  性质：最小树形图基于贪心和缩点的思想。
8  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
9
10 算法概述：
11 为了求一个图的最小树形图，1、先求出最短弧集合  $E_0$ ；2、如果  $E_0$  不存在，则图的最小树形图也不存在；3、如果  $E_0$  存
   在且不具有环，
12 则  $E_0$  就是最小树形图；4、如果  $E_0$  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  $G_1$ ，然后对  $G_1$  继续求
   其的最小树形图，
13 直到求到图  $G_i$ ，如果  $G_i$  不具有最小树形图，那么此图不存在最小树形图，如果  $G_i$  存在最小树形图，那么逐层展开，就
   得到了原图的最
14 小树形图。
15
16 模板1：
17 题意：
18     给出  $n(1 \leq n \leq 1000)$  个点， $m(1 \leq m \leq 10000)$  条边，求出最小树形图，并输出根节点，点从 0 开始
19 输入：
20 3 1
21 0 1 1
22
23 4 4
24 0 1 10
25 0 2 10
26 1 3 20
27 2 3 30
28
29 输出：
30 impossible
31
32 40 0
33
34 int n,m,pos,pre[N],id[N],vis[N];
35 //in[i]存最小入边权,pre[v]为该边的起点
36 ll in[N];
37 struct node{
38     int u,v;
39     ll w;
40 }edge[M];
41
42 ll Directed_MST(int root,int V,int E){
43     //存最小树形图总权值
44     ll ret=0;
45     while(1){
46         //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人边,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         }

```

```

106         sum++;
107         //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)
108         for(int i=m;i<n+m;i++){
109             edge[i].u=0; edge[i].v=i-m+1; edge[i].w=sum;
110         }
111         ll ans=Directed_MST(0,n+1,m+n);
112         //n+1为总结点数,m+n为总边数
113         //ans代表以超级节点0为根的最小树形图的总权值,
114         //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
115         //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
116         if(ans==-1||ans-sum>=sum) printf("impossible\n\n");
117         else printf("%lld %d\n\n",ans-sum,pos-m);
118     }
119     return 0;
120 }
121
122 模板2:
123 题意:
124     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
125     的权值之和, 如果没有以r为根的最小树形图, 输出 -1。
126
127 输入:
128 4 6 1 //n,m,root
129 1 2 3
130 1 3 1
131 4 1 2
132 4 2 2
133 3 2 1
134 3 4 1
135
136 输出:
137 3
138
139 int n,m,pos,pre[N],id[N],vis[N],root;
140 //in[i]存最小入边权,pre[v]为该边的起点
141 ll in[N];
142 struct node{
143     int u,v,w;
144 }edge[M];
145
146 ll Directed_MST(int root,int V,int E){
147     ll ret=0;
148     while(1){
149         for(int i=1;i<=V;i++) in[i]=INF;
150         for(int i=1;i<=E;i++){
151             int u=edge[i].u,v=edge[i].v;
152             if(edge[i].w<in[v]&&u!=v){
153                 in[v]=edge[i].w;
154                 pre[v]=u;
155             }
156         }
157         for(int i=1;i<=V;i++){
158             if(root==i) continue;
159             if(in[i]==INF) return -1;
160         }
161         int cnt=0;
162         memset(id,-1, sizeof(int)*(n+1));
163         memset(vis,-1, sizeof(int)*(n+1));
164         in[root]=0;

```

```

165     for(int i=1;i<=V;i++){
166         ret+=in[i];
167         int v=i;
168         while(vis[v]!=i&&id[v]==-1&&v!=root){
169             vis[v]=i;
170             v=pre[v];
171         }
172         if(v!=root&&id[v]==-1){
173             id[v]=++cnt;
174             for(int u=pre[v];u!=v;u=pre[u]) id[u]=cnt;
175         }
176     }
177     if(cnt==0) break; //无环则break
178     for(int i=1;i<=V;i++){
179         if(id[i]==-1) id[i]=++cnt;
180     }
181     ///3.建立新图,缩点,重新标记
182     for(int i=1;i<=E;i++){
183         int u=edge[i].u,v=edge[i].v;
184         edge[i].u=id[u]; edge[i].v=id[v];
185         if(id[u]!=id[v]){
186             edge[i].w=in[v];
187         }
188     }
189     V=cnt;
190     root=id[root];
191 }
192 return ret;
193 }
194
195 int main(){
196     scanf("%d%d%d",&n,&m,&root);
197     for(int i=1;i<=m;i++) scanf("%d%d%d",&edge[i].u,&edge[i].v,&edge[i].w);
198     printf("%lld\n",Directed_MST(root,n,m));
199     return 0;
200 }

```

## 4.6 割点、桥、双联通分量

### 1、割点

2 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

3

### 4 Tarjan算法

5 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

6 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

7 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

8 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（

9 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]。

10 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]=

11 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])。

12

```

13
14 代码:
15 //dfs初始为0,tot=0, head=-1
16 int n,m,head[N],tot,dfn[N],low[N],dfs,iscut[N];
17 struct node{
18     int v,nx;
19 }edge[M<<1];
20
21 void add_edge(int u,int v){
22     edge[tot].v=v;
23     edge[tot].nx=head[u];head[u]=tot++;
24 }
25
26 //求出所有割点,id为上一条边
27 void Tarjan(int u,int id){
28     int cnt=0;
29     dfn[u]=low[u]=++dfs;
30     for(int i=head[u];~i;i=edge[i].nx){
31         int v=edge[i].v;
32         if(i==(id^1)) continue;
33         if(!dfn[v]){
34             cnt++; Tarjan(v,i);
35             low[u]=min(low[u],low[v]);
36             if(low[v]>=dfn[u]) iscut[u]=1;
37         }else low[u]=min(low[u],dfn[v]);
38     }
39     if(cnt==1&&id==-1) iscut[u]=0;
40 }
41
42
43 //求一个图中去掉两个点后的最大连通块数
44 int n,m,head[N],tot,dfn[N],low[N],dfs,iscut[N];
45 struct node{
46     int v,nx;
47 }edge[M];
48
49 void add_edge(int u,int v){
50     edge[tot].v=v;
51     edge[tot].nx=head[u];
52     head[u]=tot++;
53 }
54
55 //一个点时父节点iscut=0, iscut>0时就是砍掉该点后连通块个数
56 void Tarjan(int u,int pre,int ban){
57     int cnt=0;
58     dfn[u]=low[u]=++dfs;
59     for(int i=head[u];i;i=edge[i].nx){
60         int v=edge[i].v;
61         if(v==pre||v==ban) continue;
62         if(!dfn[v]){
63             cnt++; Tarjan(v,u,ban);
64             low[u]=min(low[u],low[v]);
65             if(low[v]>=dfn[u]) iscut[u]++;
66         }else low[u]=min(low[u],dfn[v]);
67     }
68     //if(cnt==1&&pre==-1) iscut[u]=0;
69 }
70
71 int main() {

```

```

72     int x,y;
73     while(scanf("%d%d",&n,&m)==2){
74         memset(head,0,sizeof(int)*(n+1)); tot=1;
75         for(int i=1;i<=m;i++){
76             scanf("%d%d",&x,&y);
77             add_edge(x,y); add_edge(y,x);
78         }
79         int ret=0;
80         //去掉一个点后,求割点
81         for(int j=0;j<n;j++){
82             memset(dfn,0,sizeof(int)*(n+1));dfs=0;
83             //对节点进行初始化
84             for(int i=0;i<n;i++) iscut[i]=1; iscut[j]=0;
85             int k=0;
86             for(int i=0;i<n;i++){
87                 if(i==j||dfn[i]) continue;
88                 iscut[i]=0; k++;
89                 Tarjan(i,i,j);
90             }
91             for(int i=0;i<n;i++){
92                 if(i==j) continue;
93                 ret=max(ret,k+iscut[i]-1);
94             }
95         }
96         printf("%d\n",ret);
97     }
98 }
99
100 2、桥
101 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。
102     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。
103 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
104 点),所以顶点u是割点。
105 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
106 边就是割边。
107 //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
108 父节点走到当前点的边,
109 //则可以用父节点的dfn更新当前点的low。
110 //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。
111 //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
112 //dfs初始为0,tot=0, head=-1
113 int n,m,head[N],tot,dfn[N],low[N],dfs;
114 int ret;
115 struct node{
116     int v,w,nx;
117 }edge[M<<1];
118 void add_edge(int u,int v,int w){
119     edge[tot].v=v;edge[tot].w=w;
120     edge[tot].nx=head[u];head[u]=tot++;
121 }
122
123 //求出所有割点,id为上一条边
124 void Tarjan(int u,int id){
125     dfn[u]=low[u]=++dfs;
126     for(int i=head[u];~i;i=edge[i].nx){
127         int v=edge[i].v;

```

```

128         if(i==(id^1)) continue;
129         if(!dfn[v]){
130             Tarjan(v,i);
131             low[u]=min(low[u],low[v]);
132             //或low[v]>dfn[u]
133             if(low[v]==dfn[v]) {
134                 //edge[i]为桥
135                 ret=min(ret,edge[i].w);
136             }
137         }else low[u]=min(low[u],dfn[v]);
138     }
139 }
140
141 int main() {
142     int u,v,w;
143     while(scanf("%d%d",&n,&m)==2){
144         if(n==0&&m==0) break;
145         tot=0;memset(head,-1,sizeof(int)*(n+1));
146         for(int i=1;i<=m;i++){
147             scanf("%d%d%d",&u,&v,&w);
148             add_edge(u,v,w); add_edge(v,u,w);
149         }
150         int cnt=0;ret=inf;
151         memset(dfn,0,sizeof(int)*(n+1)); dfs=0;
152         for(int i=1;i<=n;i++){
153             if(!dfn[i]){
154                 cnt++; Tarjan(i,-1);
155             }
156         }
157         if(cnt>1){
158             printf("0\n");
159             continue;
160         }
161         if(ret==inf){
162             printf("-1\n");
163             continue;
164         }
165         printf("%d\n",ret==0?1:ret);
166     }
167 }

```

### 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

```

182 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

```

```

240 void Tarjan(int u,int id){
241     low[u]=dfn[u]=++dfs;
242     st.push(u);
243     instack[u]=1;
244     for(int i=head[u];~i;i=edge[i].nx){
245         int v=edge[i].v;
246         if(id==(i^1)) continue;
247         if(!dfn[v]) {
248             Tarjan(v, i);
249             low[u] = min(low[u],low[v]);
250         }else if(instack[v]&&dfn[v]<dfn[u]) low[u]=min(low[u],low[v]);
251     }
252     if(dfn[u]==low[u]){
253         blocks++;
254         int curr;
255         do{
256             curr=st.top();
257             st.pop();
258             instack[curr]=0;
259             ebc[curr]=blocks;
260         }while(curr!=u);
261     }
262 }

```

263 点双连通分量 BCC

264 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

265 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

266 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。

267 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。

268 注意:两个直接连接的点也是bcc,但是单个点不算bcc

269

270 //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;

271 //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

272

273 //bcc存的是点双连通分量中的点,初始化bcc\_cnt=-1, bccno为-1

274 vector<int>bcc[N];

275 int bcc\_cnt,bccno[N];

276 stack<node>st;

277

```

278 void Tarjan(int u,int id){
279     dfn[u]=low[u]=++dfs;
280     for(int i=head[u];~i;i=edge[i].nx){
281         int v=edge[i].v;
282         if(id==(i^1)) continue;
283         if(!dfn[v]){
284             st.push(edge[i]);
285             Tarjan(v,i);
286             low[u]=min(low[u],low[v]);
287             //判断桥
288             if(low[v]==dfn[v]) ret++;
289             int cnt=0;node x;
290             //获得点双连通分量
291             if(low[v]>=dfn[u]){
292                 bcc_cnt++; bcc[bcc_cnt].clear();
293                 do{
294                     cnt++;
295                     x=st.top(); st.pop();
296                     if(bccno[x.u]!=bcc_cnt){
297

```

```

298         bcc[bcc_cnt].push_back(x.u);
299         bccno[x.u]=bcc_cnt;
300     }
301     if(bccno[x.v]!=bcc_cnt){
302         bcc[bcc_cnt].push_back(x.v);
303         bccno[x.v]=bcc_cnt;
304     }
305     }while(x.u!=u||x.v!=v);
306     if(bcc[bcc_cnt].size()<cnt) ret2+=cnt;
307 }
308 }else if(dfn[v]<dfn[u]){ //加入还没有加入过的边
309     st.push(edge[i]);
310     low[u]=min(low[u],dfn[v]);
311 }
312 }
313 }

```

## 4.7 二分匹配

```

1  1、匈牙利算法(O(n*m))
2
3  二分图最小顶点覆盖 = 二分图最大匹配;
4  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。
5
6  DAG图的最小路径覆盖 = 节点数 (n) - 最大匹配数;
7  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。
8
9  二分图最大独立集 = 节点数 (n) - 最大匹配数;
10 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
11
12 最大匹配数: 最大匹配的匹配边的数目
13
14 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
15
16 最大独立数: 选取最多的点, 使任意所选两点均不相连
17
18 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)。
19
20 int k,n,m,girl[N];
21 vector<int>g[N];
22 bool vis[N];
23 //n表示男生数, m表示女生数, girl[i]表示i女生搭档的男生
24 bool Match(int x){
25     for(int i=0;i<g[x].size();i++){
26         int y=g[x][i];
27         if(!vis[y]){
28             vis[y]=1;
29             if(!girl[y]||Match(girl[y])){
30                 girl[y]=x; return true;
31             }
32         }
33     }
34     return false;
35 }
36
37 void gao(int n,int m){
38     int ret=0;

```

```

39     memset(girl,0, sizeof(int)*(m+1));
40     for(int i=1;i<=n;i++){
41         memset(vis,0, sizeof(bool)*(m+1));
42         if(Match(i)) ret++;
43     }
44     printf("%d\n",ret);
45 }

```

## 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

2

3 输入:

4 //n,m,s,t,k

5 3 2 1 3 1

6 1 2 1

7 2 3 2

8

```

9  int n,m,s,t,k;
10 bool vis[N];
11 ll dp[1005][N];
12 struct edge{
13     int v,nx;
14     ll w;
15 }e[M];
16 int tot,head[N];
17 void add_edge(int u,int v,ll w){
18     e[tot].v=v;e[tot].w=w;e[tot].nx=head[u];
19     head[u]=tot++;
20 }
21
22 struct node{
23     int u; ll w;
24     bool operator<(const node&t)const{
25         return w>t.w;
26     }
27 };
28
29 void Dijkstra(int st,int ed,int k){
30     memset(dp[k],0x3f, sizeof(ll)*(n+1));
31     memset(vis,0, sizeof(bool)*(n+1));
32     dp[k][st]=0;
33     priority_queue<node>q;
34     q.push(node{st,0});
35     while(!q.empty()){
36         node x=q.top();q.pop();
37         if(vis[x.u]) continue;
38         vis[x.u]=1;
39         for(int i=head[x.u];~i;i=e[i].nx){
40             int v=e[i].v; ll w=e[i].w;
41             if(vis[v]) continue;
42             if(k==0){
43                 if(dp[k][v]>dp[k][x.u]+e[i].w){
44                     dp[k][v]=dp[k][x.u]+e[i].w;
45                     q.push(node{v,dp[k][v]});
46                 }
47             }else{
48                 ll len=min(dp[k-1][x.u],dp[k][x.u]+w);

```

```

49         if(dp[k][v]>len){
50             dp[k][v]=len;
51             q.push(node{v,len});
52         }
53     }
54 }
55 }
56 }
57
58 int main(){
59     int u,v,w;
60     scanf("%d%d%d%d",&n,&m,&s,&t,&k);
61     tot=0; memset(head,-1, sizeof(int)*(n+1));
62     for(int i=1;i<=m;i++){
63         scanf("%d%d%d",&u,&v,&w);
64         add_edge(u,v,w);add_edge(v,u,w);
65     }
66     for(int i=0;i<=k;i++) Dijkstra(s,t,i);
67     ll ret=INF;
68     for(int i=0;i<=k;i++) ret=min(ret,dp[i][t]);
69     printf("%lld\n",ret);
70 }

```

## 4.9 2-SAT

1 概念:

2 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

3 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, 一般题中只要求出一种即可。

4

5 1、tarjan

6 假设有a1,a2和b1,b2两对, 已知a1和b2间有矛盾, 于是为了方案自治, 由于两者中必须选一个, 所以我们要拉两条有向边(a1,b1)和(b2,a2)表示选了a1则必须选b1,

7 选了b2则必须选a2才能够自治。然后通过这样子建边我们跑一遍Tarjan SCC判断是否有一个集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。

8 构造方案只需要把几个不矛盾的 SCC 拼起来就好了。

9

10 寻求一组可行解:

11 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。

12

13

14 2、爆搜模板, 可以求字典序可行最优解

```

15 struct Twosat {
16     int n;
17     vector<int> g[N * 2];
18     bool mark[N * 2];
19     int s[N * 2], c;
20     bool dfs(int x) {
21         if (mark[x ^ 1]) return false;
22         if (mark[x]) return true;
23         mark[x] = true;
24         s[c++] = x;
25         for (int i = 0; i < (int)g[x].size(); i++)
26             if (!dfs(g[x][i])) return false;
27         return true;
28     }

```

```

29 void init(int n) {
30     this->n = n;
31     for (int i = 0; i < n * 2; i++) g[i].clear();
32     memset(mark, 0, sizeof(mark));
33 }
34 void add_clause(int x, int y) { // 这个函数随意变化
35     g[x].push_back(y ^ 1);      // 选了 x 就必须选 y^1
36     g[y].push_back(x ^ 1);
37 }
38 bool solve() {
39     for (int i = 0; i < n * 2; i += 2)
40         if (!mark[i] && !mark[i + 1]) {
41             c = 0;
42             if (!dfs(i)) {
43                 while (c > 0) mark[s[--c]] = false;
44                 if (!dfs(i + 1)) return false;
45             }
46         }
47     return true;
48 }
49 }sat;
50
51 int main() {
52     int x,y;
53     while(scanf("%d%d",&n,&m)==2){
54         sat.init(n);
55         for(int i=1;i<=m;i++){
56             scanf("%d%d",&x,&y);
57             x--;y--;
58             sat.add_clause(x,y);
59         }
60         if(sat.solve()){
61             for(int i=0;i<2*n;i++){
62                 if(sat.mark[i]) printf("%d\n",i+1);
63             }
64             }else printf("NIE\n");
65         }
66     }
67 }

```

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 $a \vee b == 1$ , 其中a和b为真假状态, 那么有三种建边情况
  - 1)  $a=b=1 \vee 0$ , 那么会建边 $0 \rightarrow 1$ 或 $1 \rightarrow 0$
  - 2)  $a=1, b=0$ 或 $a=0, b=1$ , 那么会建边 $0 \rightarrow 0, 1 \rightarrow 1$
  - 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 $a \text{ op } b == c$ , 其中op有and, or, xor  
有六种情况需要讨论:
  - 1)  $a \text{ and } b == 1$ , 则需建边 $a=0 \rightarrow a=1, b=0 \rightarrow b=1$ 。这里需要体会一下, 只需要两条即可
  - 2)  $a \text{ and } b == 0$ , 则需建边 $a=1 \rightarrow b=0, b=1 \rightarrow a=0$ 。
  - 3)  $a \text{ or } b == 1$ , 则需建边 $a=0 \rightarrow b=1, b=0 \rightarrow a=1$ 。
  - 4)  $a \text{ or } b == 0$ , 则需建边 $a=1 \rightarrow a=0, b=1 \rightarrow b=0$ 。同1情况
  - 5)  $a \text{ xor } b == 1$ , 则需建边 $a=0 \rightarrow b=1, a=1 \rightarrow b=0, b=0 \rightarrow a=1, b=1 \rightarrow a=0$ 。
  - 6)  $a \text{ xor } b == 0$ , 则需建边 $a=0 \rightarrow b=0, b=0 \rightarrow a=0, a=1 \rightarrow b=1, b=1 \rightarrow a=1$ 。

87 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  
 88 于  
 89 有两种情况需要讨论:  
 89 1)  $a=1 \rightarrow b=0, c=0$   $b=1, c=1 \rightarrow a=0$   
 90 2)  $a=0 \rightarrow b=1, b=0 \rightarrow a=1$   
 91  
 92 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  
 93 二分答案, 通过二分的数据进行约束, 创建2-sat  
 94  
 95 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,  
 96 某些牛棚不能连在同一个S。  
 96 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。  
 97 二分答案, 用2-sat判断是否可行  
 98 1.hate关系的a,b。  $a \rightarrow b^1, b \rightarrow a^1, a^1 \rightarrow b, b^1 \rightarrow a$   
 99 2.friend关系的a,b。  $a \rightarrow b, b \rightarrow a, a^1 \rightarrow b^1, b^1 \rightarrow a^1$   
 100 接下来的也要检查, 因为引入参数, 就是多了约束条件了  
 101 这四种情况就是i, j到达对方的所有情况了  
 102 3. $dis[a]+dis[b]>limit$   $a \rightarrow b^1, b \rightarrow a^1$   
 103 4. $dis[a^1]+dis[b^1]>limit$   $a^1 \rightarrow b, b^1 \rightarrow a$   
 104 5. $dis[a]+dis[b^1]+tdis>limit$   $a \rightarrow b, b^1 \rightarrow a^1$   
 105 6. $dis[a^1]+dis[b]+tdis>limit$   $a^1 \rightarrow b^1, b \rightarrow a$   
 106  
 107 7、两者 (A, B) 要么都取, 要么都不取  
 108 建边:  $a \rightarrow b, b \rightarrow a, a^1 \rightarrow b^1, b^1 \rightarrow a^1$   
 109  
 110 8、两者 (A, A') 必取A  
 111 建边:  $a^1 \rightarrow a$   
 112

#### 4.10 LCA

1 LCA:  
 2 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  
 3 大的  
 4 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。  
 5 1、Tarjan离线算法  
 6 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 $O(n+m)$ 。  
 7 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:  
 8 1) 选一个点为根节点, 从根节点开始。  
 9 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。  
 10 3) 若是v还有子节点, 返回2, 否则下一步。  
 11 4) 合并v到u上。  
 12 5) 寻找与当前点u有询问关系的点v。  
 13 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。  
 14  
 15 代码:  
 16 //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。  
 17 //输出m次询问的公共祖先  
 18 //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  
 19 //注意并查集parent需要初始化  
 20 `int n,m,s,ans[N],parent[N];`  
 21 `int ehead[N],etot;`  
 22 `struct edge{`  
 23 `int v,nx;`  
 24 `}e[N<<1];`  
 25

```

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",&n,&m,&s);
74     etot=qtot=1;
75     for(int i=1;i<n;i++){
76         scanf("%d",&u,&v);
77         add_edge(u,v); add_edge(v,u);
78     }
79     for(int i=1;i<=m;i++){
80         scanf("%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;

```



```

85     Tarjan(s);
86     for(int i=1;i<=m;i++) printf("%d\n",ans[i]);
87 }
88
89 2、树上倍增LCA
90
91 const int MAX_DEP = 20;
92
93 // 倍增 $2^k$ 的父亲
94 int fa[N][MAX_DEP];
95 int dep[N];
96
97 int n,m,s;
98 int head[N],tot=1;
99 struct edge{
100     int v,nx;
101 }e[N<<1];
102
103 void add_edge(int u,int v){
104     e[tot].v=v; e[tot].nx=head[u];
105     head[u]=tot++;
106 }
107
108 void lineFa(int u,int v){
109     fa[u][0]=v;
110     for(int i=1;i<MAX_DEP;i++)
111         v=fa[u][i]=fa[v][i-1];
112 }
113
114 void dfs(int u,int pre){
115     for(int i=head[u];i;i=e[i].nx){
116         int v=e[i].v;
117         if(v==pre) continue;
118         dep[v]=dep[u]+1;
119         lineFa(v,u);
120         dfs(v,u);
121     }
122 }
123
124 int LCA(int u,int v){
125     if(dep[u]>dep[v]) swap(u,v);
126     int hu=dep[u],hv=dep[v];
127     int tu=u,tv=v;
128     for(int det=hv-hu,i=0;det;det>>=1,i++){
129         if(det&1) tv=fa[tv][i];
130     }
131     if(tu==tv) return tu;
132     for(int i=MAX_DEP-1;i>=0;i--){
133         if (fa[tu][i] == fa[tv][i]) {
134             continue;
135         }
136         tu = fa[tu][i];
137         tv = fa[tv][i];
138     }
139     return fa[tu][0];
140 }
141
142 int main() {
143     int u,v;

```

```

144     scanf("%d%d%d",&n,&m,&s);
145     tot=1;
146     for(int i=1;i<n;i++){
147         scanf("%d%d",&u,&v);
148         add_edge(u,v); add_edge(v,u);
149     }
150     dep[s]=0;
151     dfs(s,0);
152     for(int i=1;i<=m;i++){
153         scanf("%d%d",&u,&v);
154         printf("%d\n",LCA(u,v));
155     }
156 }
157
158 3、st表查询
159 //注意调用LCA_init初始化
160 int n,m,rt,head[N],tot;
161 int dfn[N],pos[N],rmq[N],dno;
162 struct edge{
163     int v,nx;
164 }e[N<<1];
165 void add_edge(int u,int v){
166     e[tot].v=v; e[tot].nx=head[u];
167     head[u]=tot++;
168 }
169
170 struct ST {
171     int k2[21], st[21][N], Log[N];
172     void init_st(int n) {
173         k2[0] = 1;
174         for (int i = 1; i <= 20; i++) k2[i] = 2 * k2[i - 1];
175         Log[0] = -1; for (int i = 1; i <= n; i++) Log[i] = Log[i / 2] + 1;
176         for (int i = 1; i <= n; i++) st[0][i] = i;
177         for (int i = 1; i <= Log[n]; i++) {
178             for (int j = 1; j + k2[i] - 1 <= n; j++) {
179                 st[i][j] = (rmq[st[i - 1][j]]<rmq[st[i - 1][j + k2[i - 1]]]) ?
180                     st[i - 1][j]:st[i - 1][j + k2[i - 1]];
181             }
182         }
183     }
184     int query_min(int x, int y) {
185         int len = log2(y - x + 1);
186         return (rmq[st[len][x]]<rmq[st[len][y - k2[len] + 1]]) ?
187             st[len][x]:st[len][y - k2[len] + 1];
188     }
189 }st;
190
191 void dfs(int u,int pre,int dep){
192     dfn[++dno]=u;
193     rmq[dno]=dep;
194     pos[u]=dno;
195     for(int i=head[u];i;i=e[i].nx){
196         int v=e[i].v;
197         if(v==pre) continue;
198         dfs(v,u,dep+1);
199         dfn[++dno]=u;
200         rmq[dno]=dep;
201     }
202 }

```

```

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",&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

6 定义:

7 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。

8 欧拉回路: 欧拉回路是指起点和终点相同的欧拉路。

9

10 无向图是否具有欧拉路或回路的判定:

11 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

12 拉路的起点和终点, 否则可取任意一点作为起点。

13 欧拉回路: 图连通, 图中所有节点度均为偶数

14

15 有向图是否具有欧拉路或回路的判定:

16 欧拉路: 图连通, 除2个端点外其余节点入度=出度; 1个端点入度比出度大1; 一个端点入度比出度小1, 取出度大者为起点, 入度大者为终点。

17 或 所有节点入度等于出度

18 欧拉回路: 图连通, 所有节点入度等于出度

19

20 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入度相同。从任意一个起始点v开始遍历, 直到再次到达点v,

21 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, 由于其出度和入度相同, 故u一定存在一条出边, 所以一定可以到达v。

22 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。

23

24

25 //有向图欧拉回路打印路径模板

26 //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

27 void dfs(int u){

28 for(int i=head[u];~i;i=e[i].nx){

29 int v=e[i].v;

30 if(vis[i]) continue;

31 vis[i]=1;

32 head[u]=i;

33 dfs(v);

34 i=head[u];

35 }

36 st[++st[0]]=u;

37 }

38

39 //非递归版

40 int st[N];

41 int syst[N\*10], systop;

42

43 void dfs(int u){

44 systop=0;

45 syst[++systop]=1; //初始点进入

46 while(systop>0){

47 int x=syst[systop], i=head[x];

48 while((~i)&&vis[i]) i=e[i].nx;

49 if(~i){

50 syst[++systop]=e[i].v;

51 vis[i]=1;

52 head[x]=e[i].nx;

53 }else{

54 systop--, st[++st[0]]=x;

55 }

56 }

57 }

58

59 //无向图欧拉回路打印路径模板, 边或点

```

60 void dfs(int u){
61     for(int i=head[u];~i;i=e[i].nx){
62         int v=e[i].v;
63         if(vis[i]) continue;
64         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
65         dfs(v);
66         //st里边存边，外部存点，st[st[0]]->st[1]输出就是路径结果
67     }
68     st[++st[0]]=u;
69 }

```

73 问题一：

74 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。

75 考虑无向图欧拉路三种情况：

- 76 1、若只存在一个点，则答案贡献为0，
- 77 2、若奇数入度点为0，则ans+=1
- 78 3、若奇数入度点为cnt，则ans+=cnt/2

80 问题二：

81 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

82 思路：

83 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

84 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

85 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

86 该条件才能存在合法序列。

87 注意：

- 88 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，
- 89 那么首字母为out++，尾字母为in++

## 5 数学

### 5.1 BM

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  #define rep(i,a,n) for (int i=a;i<n;i++)
4  #define per(i,a,n) for (int i=n-1;i>=a;i--)
5  #define pb push_back
6  #define mp make_pair
7  #define all(x) (x).begin(),(x).end()
8  #define fi first
9  #define se second
10 #define SZ(x) ((int)(x).size())
11 typedef vector<int> VI;
12 typedef long long ll;
13 typedef pair<int,int> PII;
14 const ll mod=1000000007;
15 ll powmod(ll a,ll b) {ll res=1;a%=mod; for(;;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}
16     return res;}
17 // head
18 ll n;
19 namespace linear_seq {
20     const int N=10010;
21     ll res[N],base[N],_c[N],_md[N];
22
23     vector<int> Md;
24     void mul(ll *a,ll *b,int k) {
25         rep(i,0,k+k) _c[i]=0;
26         rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
27         for (int i=k+k-1;i>=k;i--) if (_c[i])
28             rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
29         rep(i,0,k) a[i]=_c[i];
30     }
31     int solve(ll n,VI a,VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
32         ll ans=0,pnt=0;
33         int k=SZ(a);
34         assert(SZ(a)==SZ(b));
35         rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
36         Md.clear();
37         rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
38         rep(i,0,k) res[i]=base[i]=0;
39         res[0]=1;
40         while ((1ll<pnt)<=n) pnt++;
41         for (int p=pnt;p>=0;p--) {
42             mul(res,res,k);
43             if ((n>p)&1) {
44                 for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
45                 rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
46             }
47         }
48         rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
49         if (ans<0) ans+=mod;
50         return ans;
51     }
52     VI BM(VI s) {
53         VI C(1,1),B(1,1);
54         int L=0,m=1,b=1;

```

```

55     rep(n,0,SZ(s)) {
56         ll d=0;
57         rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
58         if (d==0) ++m;
59         else if (2*L<=n) {
60             VI T=C;
61             ll c=mod-d*powmod(b,mod-2)%mod;
62             while (SZ(C)<SZ(B)+m) C.pb(0);
63             rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
64             L=n+1-L; B=T; b=d; m=1;
65         } else {
66             ll c=mod-d*powmod(b,mod-2)%mod;
67             while (SZ(C)<SZ(B)+m) C.pb(0);
68             rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
69             ++m;
70         }
71     }
72     return C;
73 }
74 int gao(VI a,ll n) {
75     VI c=BM(a);
76     c.erase(c.begin());
77     rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
78     return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
79 }
80 };
81
82 int main() {
83     /*push_back 进去前 8~10 项左右、最后调用 gao 得第 n 项*/
84     vector<int>v;
85     v.push_back(3);
86     v.push_back(9);
87     v.push_back(20);
88     v.push_back(46);
89     v.push_back(106);
90     v.push_back(244);
91     v.push_back(560);
92     v.push_back(1286);
93     v.push_back(2956);
94     v.push_back(6794);
95     int nCase;
96     scanf("%d", &nCase);
97     while(nCase--){
98         scanf("%lld", &n);
99         printf("%lld\n",1LL * linear_seq::gao(v,n-1) % mod);
100     }
101 }

```

## 5.2 gcd、ex-gcd

```

1 1、gcd
2 ll gcd(ll a,ll b){
3     return b>0?gcd(b,a%b):a;
4 }
5
6 斐波那契数列的最大公约数定理: gcd(F(m),F(n))=F(gcd(m,n))
7 F[0]=0,F[1]=1...
8

```

```

9 2、ex-gcd
10 void extend_gcd(ll a, ll b, ll &x, ll &y) {
11     if (!b){
12         x = 1, y = 0;
13         return;
14     }
15     else{
16         extend_gcd(b, a % b, y, x);
17         y -= x * (a / b);
18         return;
19     }
20 }
21
22 ll inv(ll a, ll n) {
23     ll x, y;
24     extend_gcd(a,n,x,y);
25     x = (x % n + n) % n;
26     return x;
27 }

```

### 5.3 拉格朗日插值法

```

1  /*
2  测试用例 (函数)  $x^3-2x+7$ 
3  a[]={7,6,11,28,63}
4  第0-4项和 115
5  最高次项3, 代入项数4
6  */
7
8  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; }
9
10 namespace polysum {
11     #define rep(i,a,n) for (int i=a;i<n;i++)
12     #define per(i,a,n) for (int i=n-1;i>=a;i--)
13     const int D = 1e5;
14     ll a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
15     //函数用途: 给出数列的 (d+1) 项, 其中d为最高次方项
16     //求出数列的第n项, 数组下标从0开始
17     ll calcn(int d, ll* a, ll n) { // a[0].. a[d] a[n]
18         if (n <= d) return a[n];
19         p1[0] = p2[0] = 1;
20         rep(i, 0, d + 1) {
21             ll t = (n - i + mod) % mod;
22             p1[i + 1] = p1[i] * t % mod;
23         }
24         rep(i, 0, d + 1) {
25             ll t = (n - d + i + mod) % mod;
26             p2[i + 1] = p2[i] * t % mod;
27         }
28         ll ans = 0;
29         rep(i, 0, d + 1) {
30             ll t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
31             if ((d - i) & 1) ans = (ans - t + mod) % mod;
32             else ans = (ans + t) % mod;
33         }
34         return ans;
35     }

```



```

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[0];
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]
63 * t) % mod + mod) % mod;
64         else p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
65     }
66     c = powmod(p4, mod - 2) * (mod - p3) % mod;
67     rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
68     rep(i, 0, m + 2) C[i] = h[i][0];
69     ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
70     if (ans < 0) ans += mod;
71     return ans;
72 } // polysum::init();
73
74 ll b[N];
75 int n, m;
76 int l, r;
77
78 int main(){
79     int t;
80     scanf("%d", &t);
81     while(t--){
82         scanf("%d%d", &n, &m);
83         polysum::init(n+10);
84         for(int i=0; i<=n; i++)
85             scanf("%lld", &b[i]);
86         while(m--){
87             scanf("%d%d", &l, &r);
88             printf("%lld\n", ((polysum::polysum(n, b, r+1) - polysum::polysum(n, b, l))%mod+
89 mod)%mod);
90         }
91     }
92 }

```

## 5.4 素数

```

1 1、素数筛
2 //Mark中标记为true的为合数
3 int prime[N];
4 bool Mark[N];
5 int cnt=0;
6
7 void Prime(int n){
8     for(int i=2;i<=n;i++){
9         if(Mark[i]==0)
10             prime[cnt++]=i;
11         for(int j=0;j<cnt&&prime[j]*i<=n;j++){
12             Mark[i*prime[j]]=1;
13             if(i%prime[j]==0)
14                 break;
15         }
16     }
17 }
18
19 //获得[2,n]的所有最小质因子和, 1不为质因子
20 ll res=0;
21 int n,cnt;
22 int prime[N];
23 bool Mark[N];
24
25 void Prime(int n){
26     for(int i=2;i<=n;i++){
27         if(!Mark[i])
28             res+=prime[cnt++]=i;
29         for(int j=0,e=n/i;j<cnt&&prime[j]<=e;j++){
30             Mark[i*prime[j]]=1;
31             res+=prime[j];
32             if(i%prime[j]==0) break;
33         }
34     }
35 }
36
37 //求[2,n]内所有数能分解出的合数因子,复杂度nlogn
38 void init(int n){
39     Prime(n);
40     tot=0; memset(head,-1, sizeof(head));
41     for(int v=2;v<=n;v++){
42         if(!Mark[v]) continue; //若v为合数则可以加入答案
43         for(int u=v;u<=n;u+=v){
44             add_edge(u,v);
45         }
46     }
47 }
48
49 //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,  $1*6, 6*1, 2*3, 3*2$ 
50 //求[1,n]之间 $f(i)$ 的和, 其中 $n \leq 2e7$ 
51 void Prime(int n){
52     dp[1]=1; //必须放在第一个
53     for(int i=2;i<=n;i++){
54         if(Mark[i]==0) {
55             prime[cnt++]=i; dp[i]=2;
56         }

```

```

57     for(int j=0;j<cnt&&prime[j]*i<=n;j++){
58         int di=i*prime[j];
59         Mark[di]=1;
60         if(i%prime[j]) dp[di]=dp[i]*2;
61         else{
62             if((i/prime[j])%prime[j]==0) dp[di]=0;
63             else dp[di]=dp[i/prime[j]];
64             break;
65         }
66     }
67 }
68 for(int i=2;i<=n;i++) dp[i]+=dp[i-1];
69 }
70
71 2、单点判断
72 bool isPrime(ll num){
73     if (num == 2 || num == 3)
74         return true;
75     if (num % 6 != 1 && num % 6 != 5)
76         return false;
77     for (ll i = 5; i*i <= num; i += 6)
78         if (num % i == 0 || num % (i+2) == 0)
79             return false;
80     return true;
81 }
82
83
84 3、一个素数为P,它之前的素数为Q,求Q!%P的值
85 int cnt=0;
86 ll p,q;
87
88 bool isPrime(ll num){
89     if (num == 2 || num == 3)
90         return true;
91     if (num % 6 != 1 && num % 6 != 5)
92         return false;
93     for (ll i = 5; i*i <= num; i += 6)
94         if (num % i == 0 || num % (i+2) == 0)
95             return false;
96     return true;
97 }
98
99 void extend_gcd(ll a, ll b, ll &x, ll &y) {
100     if (!b){
101         x = 1, y = 0;
102         return;
103     }
104     else{
105         extend_gcd(b, a % b, y, x);
106         y -= x * (a / b);
107         return;
108     }
109 }
110
111 ll inv(ll a, ll n) {
112     ll x, y;
113     extend_gcd(a,n,x,y);
114     x = (x % n + n) % n;
115     return x;

```

```

116 }
117
118 int main() {
119     int t;
120     scanf("%d",&t);
121     while(t--){
122         scanf("%lld",&p);
123         q=p-1;
124         while(!isPrime(q))
125             q--;
126         ll top = p-q-1;
127         if(top==0)
128             top=1;
129         ll res=1;
130         for(ll i=1;i<=top;i++){
131             res=res*i%p;
132         }
133         res = inv(res,p);
134         printf("%lld\n",res);
135     }
136 }
137
138 4、
139 LL Mult_Mod(LL a,LL b,LL m)//res=(a*b)%m
140 {
141     a%=m;
142     b%=m;
143     LL res=0;
144     while(b)
145     {
146         if(b&1)
147             res=(res+a)%m;
148         a=(a<<=1)%m;
149         b>>=1;
150     }
151     return res%m;
152 }
153 LL Pow_Mod(LL a, LL b, LL m)//res=(a^b)%m
154 {
155     LL res=1;
156     LL k=a;
157     while(b)
158     {
159         if((b&1))
160             res=Mult_Mod(res,k,m)%m;
161
162         k=Mult_Mod(k,k,m)%m;
163         b>>=1;
164     }
165     return res%m;
166 }
167
168 bool Witness(LL a,LL n,LL x,LL sum)
169 {
170     LL judge=Pow_Mod(a,x,n);
171     if(judge==n-1||judge==1)
172         return 1;
173
174     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     }

```

```

234         if(i==k)
235         {
236             y=x;
237             k<=1;
238         }
239     }
240 }
241
242 int total;//因子的个数
243 LL factor[N];//存储所有因子的数组，无序的
244 void Find_fac(LL n)//对n进行素因子分解，存入factor
245 {
246     if(Miller_Rabin(n))//是素数就把这个素因子存起来
247     {
248         factor[++total]=n;
249         return;
250     }
251
252     long long p=n;
253     while(p>=n)//值变化，防止陷入死循环k
254         p=Pollard_Rho(p,rand()%(n-1)+1);
255
256     Find_fac(n/p);
257     Find_fac(p);
258 }

```

## 5.5 高斯消元

```

1  1.解同余方程模板
2  ll a[N][N],x[N];
3  inline ll gcd(ll a,ll b) {
4      return b ? gcd(b, a % b) : a;
5  }
6
7  inline ll lcm(ll a,ll b) {
8      return a/gcd(a,b)*b;
9  }
10
11 ll inv(ll a,ll p){
12     if(a == 1) return 1;
13     return inv(p%a,p)*(p-p/a)%p;
14 }
15
16 int Gauss(int equ,int var) {
17     int max_r, col, k;
18     for (k = 0, col = 0; k < equ && col < var; k++, col++) {
19         max_r = k;
20         for(int i = k+1; i < equ;i++)
21             if(abs(a[i][col]) > abs(a[max_r][col]))
22                 max_r = i;
23         if(a[max_r][col] == 0){
24             k--;
25             continue;
26         }
27         if(max_r != k)
28             for(int j = col; j < var+1;j++)
29                 swap(a[k][j],a[max_r][j]);
30         for(int i = k+1;i < equ;i++) {

```

```

31         if (a[i][col] != 0) {
32             ll LCM = lcm(abs(a[i][col]),abs(a[k][col]));
33             ll ta = LCM/abs(a[i][col]);
34             ll tb = LCM/abs(a[k][col]);
35             if(a[i][col]*a[k][col] < 0)tb = -tb;
36             for(int j = col;j < var+1;j++)
37                 a[i][j] = ((a[i][j]*ta - a[k][j]*tb)%mod + mod)%mod;
38         }
39     }
40 }
41 for(int i = k;i < equ;i++)
42     if(a[i][col] != 0)
43         return -1;//无解
44 if(k < var) return var-k;//多解
45 for(int i = var-1;i >= 0;i--){
46     ll temp = a[i][var];
47     for(int j = i+1; j < var;j++) {
48         if (a[i][j] != 0) {
49             temp -= a[i][j] * x[j];
50             temp = (temp % mod + mod) % mod;
51         }
52     }
53     x[i] = (temp*inv(a[i][i],mod))%mod;
54 }
55 return 0;
56 }
57
58 2.
59 const double EPS = 1e-9;
60 inline int sign(double x){return (x>EPS)-(x<-EPS);}
61 double A[250][250];
62
63 bool gauss(int n){
64     int i,j,k,r;
65     for(i=0;i<n;i++){
66         //选一行与r与第i行交换, 提高数据值的稳定性
67         r=i;
68         for(j=i+1;j<n;j++)
69             if(fabs(A[j][i]) > fabs(A[r][i]))r=j;
70         if(r!=i)for(j=0;j<=n;j++)swap(A[r][j],A[i][j]);
71         //i行与i+1~n行消元
72         /* for(k=i+1;k<n;k++){ //从小到大消元, 中间变量f会有损失
73             double f=A[k][i]/A[i][i];
74             for(j=i;j<=n;j++)A[k][j]-=f*A[i][j];
75         }*/
76         for(j=n;j>=i;j--){ //从大到小消元, 精度更高
77             for(k=i+1;k<n;k++)
78                 A[k][j]-=A[k][i]/A[i][i]*A[i][j];
79         }
80     }
81     //判断方程时候有解
82     for(i=0;i<n;i++)if(sign(A[i][i])==0)return 0;
83     //回代过程
84     for(i=n-1;i>=0;i--){
85         for(j=i+1;j<n;j++)
86             A[i][n]-=A[j][n]*A[i][j];
87         A[i][n]/=A[i][i];
88     }
89     return 1;

```

90 }

## 5.6 几何基础模板

```

1  class Point{
2  public:
3      double x, y;
4      Point(double x = 0, double y = 0) :x(x), y(y) {}
5      Point operator + (Point a){
6          return Point(x + a.x, y + a.y);
7      }
8      Point operator - (Point a){
9          return Point(x - a.x, y - a.y);
10     }
11     bool operator < (const Point& a) const{
12         if (x == a.x) return y < a.y;
13         return x < a.x;
14     }
15     bool operator == (Point a){
16         if (x == a.x && y == a.y) return true;
17         return false;
18     }
19     double abs(void){
20         return sqrt(x * x + y * y);
21     }
22 };
23
24 typedef Point Vector;
25
26 //叉积
27 double cross(Vector a, Vector b){
28     return a.x * b.y - a.y * b.x;
29 }
30
31 //点积
32 double dot(Vector a, Vector b){
33     return a.x * b.x + a.y * b.y;
34 }
35
36 //判断方向
37 bool isclock(Point p0, Point p1, Point p2){
38     Vector a = p1 - p0;
39     Vector b = p2 - p0;
40     if (cross(a, b) < 0) return true;
41     return false;
42 }
43
44 typedef vector<Point> Polygon;
45
46 //求凸包
47 Polygon andrewScan(Polygon s) {
48     Polygon u, l;
49     if (s.size() < 3) return s;
50     sort(s.begin(), s.end());
51     u.push_back(s[0]);
52     u.push_back(s[1]);
53     l.push_back(s[s.size() - 1]);
54     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++){

```

```

111     Point &y = s[i], &z = s[(i+1)%n];
112     sum += fabs(cross(y-x, z-x))/2;
113 }
114 return fabs(sum-area) <= eps;
115 }
116
117 已知平面三个点, 求外接圆圆心
118 double x = ((y2-y1)*(y3*y3-y1*y1+x3*x3-x1*x1) - (y3-y1)*(y2*y2-y1*y1+x2*x2-x1*x1)) / (2*(x3-
119     x1)*(y2-y1) - 2*((x2-x1)*(y3-y1)));
120 double y = ((x2-x1)*(x3*x3-x1*x1+y3*y3-y1*y1) - (x3-x1)*(x2*x2-x1*x1+y2*y2-y1*y1)) / (2*(y3-
121     y1)*(x2-x1) - 2*((y2-y1)*(x3-x1)));
122 printf("%.3lf %.3lf\n", x, y);
123
124 如何判断三角形是钝角、直角、还是锐角三角形
125 1、设c为最长的边
126     若 $a^2+b^2 < c^2$ , 则为钝角三角形
127     若 $a^2+b^2 = c^2$ , 则为直角三角形
128     若 $a^2+b^2 > c^2$ , 则为锐角三角形
129 2、三点中, 若存在点乘 $<0$ , 则说明存在钝角
130 //i为顶点
131 bool check(int i, int j, int k){
132     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;
133 }
134
135 点乘是向量的内积, 叉乘是向量的外积
136 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。
137 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。
138
139 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
140
141 向量的点乘:  $a \cdot b$ ,  $x1*x2+y1*y2$ 
142 公式:  $a \cdot b = |a| \cdot |b| \cdot \cos\theta$  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
143 度的乘积;
144 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。
145
146 向量的叉乘:  $a \wedge b$ ,  $x1*y2-x2*y1$ 
147  $a \wedge b = |a| \cdot |b| \cdot \sin\theta$  向量积被定义为: 模长: (在这里 $\theta$ 表示两向量之间的夹角(共起点的前提下) ( $0^\circ \leq \theta \leq 180^\circ$ ),
148 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
149 定则。
150 (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
151 以不超过180度的转角转向b时,
152 竖起的大拇指指向是c的方向。  $c = a \wedge b$ )

```

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

```

1 0、快速幂
2 ll quick(ll a, ll b){
3     ll ret=1; a%=mod;
4     while(b){
5         if(b&1) ret=ret*a%mod;
6         b>>=1;
7         a=a*a%mod;

```

```

8     }
9     return ret;
10 }
11
12 1、求phi模板
13 ll phi(ll m) {
14     ll ans = 1;
15     for (ll i = 2; i*i <= m; i++) {
16         if (m%i == 0) {
17             m /= i;
18             ans *= i - 1;
19             while (m%i == 0) {
20                 m /= i;
21                 ans *= i;
22             }
23         }
24     }
25     if (m > 1) ans *= m - 1;
26     return ans;
27 }
28
29 2、大数取模模板
30 ll Mod(string a,ll b){
31     ll len=a.length()-1;
32     ll ans=0;
33     for(int i=0;i<=len;i++){
34         ans=(ans*10+(a[i]-'0')%b)%b;
35     }
36     return ans;
37 }
38
39 3、大数相乘模板
40 string Mul(string s,int x){
41     reverse(s.begin(),s.end());
42     int cmp=0;
43     for(int i=0;i<s.size();i++){
44         cmp=(s[i]-'0')*x+cmp;
45         s[i]=(cmp%10+'0');
46         cmp/=10;
47     }
48     while(cmp){
49         s+=(cmp%10+'0');
50         cmp/=10;
51     }
52     reverse(s.begin(),s.end());
53     return s;
54 }
55
56 4、大数相加模板:
57 string sum(string s1,string s2){
58     if(s1.length()<s2.length()) swap(s1,s2);
59     int i,j;
60     for(i=s1.length()-1,j=s2.length()-1;i>=0;i--,j--){
61         s1[i]=char(s1[i]+(j>=0?s2[j]-'0':0)); //注意细节
62         if(s1[i]-'0'>=10){
63             s1[i]=char((s1[i]-'0')%10+'0');
64             if(i) s1[i-1]++;
65             else s1='1'+s1;
66         }
67     }
68 }

```

```

67     return s1;
68 }
69
70 5、__int128 2^128次使用,只能在Linux下使用
71 inline __int128 read() {
72     __int128 x = 0, f = 1;
73     char ch = getchar();
74     while (ch < '0' || ch > '9') {
75         if (ch == '-')
76             f = -1;
77         ch = getchar();
78     }
79     while (ch >= '0' && ch <= '9') {
80         x = x * 10 + ch - '0';
81         ch = getchar();
82     }
83     return x * f;
84 }
85
86 inline void print(__int128 x) {
87     if (x < 0) {
88         putchar('-');
89         x = -x;
90     }
91     if (x > 9)
92         print(x / 10);
93     putchar(x % 10 + '0');
94 }
95
96 其他时候可以使用printf,cin,cout
97
98
99 6、快速乘
100 //O(1)快速乘
101 inline LL quick_mul(LL x,LL y,LL MOD){
102     x=x%MOD,y=y%MOD;
103     return ((x*y-(LL)(((long double)x*y+0.5)/MOD)*MOD)%MOD+MOD)%MOD;
104 }
105 //O(log)快速乘
106 inline LL quick_mul(LL a,LL n,LL m){
107     LL ans=0;
108     while(n){
109         if(n&1) ans=(ans+a)%m;
110         a=(a<<1)%m;
111         n>>=1;
112     }
113     return ans;
114 }
115
116 7、高精度相加
117 struct BigInteger {
118     static const int BASE = 10000; //高进制
119     static const int WIDTH = 4; //高进制位数
120     vector<int> s;
121     BigInteger() {}
122     BigInteger(long long num) { // 构造函数
123         *this = num;
124     }
125     //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

```

```

13 ll quick(ll a, ll b) {
14     ll res = 1;
15     while (b) {
16         if (b & 1) res = res * a % mod;
17         b >>= 1;
18         a = a * a % mod;
19     }
20     return res;
21 }
22
23 void init(int n) {
24     fac[0] = 1, fac[1] = 1;
25     for (int i = 2; i <= n; i++) {
26         fac[i] = fac[i - 1] * i % mod;
27     }
28     inv[n] = quick(fac[n], mod - 2);
29     for (int i = n - 1; i >= 0; i--) inv[i] = inv[i + 1] * (i + 1) % mod;
30 }
31
32 //n大
33 ll C(int n, int m) {
34     return fac[n] * inv[m] % mod * inv[n - m] % mod;
35 }

```

## 5.9 快速阶乘

```

1 //minamoto
2 #include<bits/stdc++.h>
3 #define R register
4 #define ll long long
5 #define fp(i,a,b) for(R int i=(a),I=(b)+1;i<I;++i)
6 #define fd(i,a,b) for(R int i=(a),I=(b)-1;i>I;--i)
7 #define go(u) for(int i=head[u],v=e[i].v;i;i=e[i].nx,v=e[i].v)
8 using namespace std;
9 const int N=(1<<17)+5;int P;
10 inline int add(R int x,R int y){return 0ll+x+y>=P?0ll+x+y-P:x+y;}
11 inline int dec(R int x,R int y){return x-y<0?x-y+P:x-y;}
12 inline int mul(R int x,R int y){return 1ll*x*y-1ll*x*y/P*P;}
13 int ksm(R int x,R int y){
14     R int res=1;
15     for(;y;y>>=1,x=mul(x,x))(y&1)?res=mul(res,x):0;
16     return res;
17 }
18 const double Pi=acos(-1.0);
19 struct cp{
20     double x,y;
21     inline cp(){}
22     inline cp(R double xx,R double yy):x(xx),y(yy){}
23     inline cp operator +(const cp &b)const{return cp(x+b.x,y+b.y);}
24     inline cp operator -(const cp &b)const{return cp(x-b.x,y-b.y);}
25     inline cp operator *(const cp &b)const{return cp(x*b.x-y*b.y,x*b.y+y*b.x);}
26     inline cp operator *(const double &b)const{return cp(x*b,y*b);}
27     inline cp operator ~( )const{return cp(x,-y);}
28 }w[2][N];
29 int r[21][N],ifac[N],lg[N],inv[N];double iv[21];
30 void Pre(){
31     iv[0]=1;
32     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[
67     i].y+0.5)))%P;
68 }
69 void calc(int *a,int *b,int n,int k){
70     static int f[N],g[N],h[N],sum[N],isum[N];
71     int len=1;while(len<=n+n)len<=1;
72     fp(i,0,n)f[i]=mul(a[i],mul(ifac[i],ifac[n-i]));
73     for(R int i=n-1;i>=0;i-=2)f[i]=P-f[i];
74     int t=dec(k,n);
75     fp(i,0,n+n)g[i]=add(i,t);
76     sum[0]=g[0];fp(i,1,n+n)sum[i]=mul(sum[i-1],g[i]);
77     isum[n+n]=ksm(sum[n+n],P-2);
78     fd(i,n+n,1)isum[i-1]=mul(isum[i],g[i]);
79     fp(i,1,n+n)g[i]=mul(isum[i],sum[i-1]);g[0]=isum[0];
80     fp(i,n+1,len-1)f[i]=0;fp(i,n+n+1,len-1)g[i]=0;
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>=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             pl=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",&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

```



```

12 const int N = (1 << 20);
13 using Complex = complex<double>;
14
15 void change(Complex p[], int n) {
16     int k = calc(n);
17     n = 1 << k;
18     vector<int> r(n, 0);
19     for (int i = 0; i < n; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (k - 1));
20     for (int i = 0; i < n; i++) if (i < r[i]) swap(p[i], p[r[i]]);
21 }
22
23 void FFT(Complex p[], int n, int type) {
24     change(p, n);
25     for (int mid = 1; mid < n; mid <= 1) { //待合并区间的长度的一半
26         Complex wn(cos(pi / mid), type * sin(pi / mid)); //单位根
27         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
28             Complex w(1, 0); //幂
29             for (int k = 0; k < mid; k++, w = w * wn) { //枚举左半部分
30                 Complex x = p[j + k], y = w * p[j + mid + k]; //蝴蝶效应
31                 p[j + k] = x + y;
32                 p[j + mid + k] = x - y;
33             }
34         }
35     }
36 }
37
38 1.大数相乘
39 给出两个n位10进制整数x和y, 你需要计算x*y。
40 int n, rev[N];
41 char x[N], y[N];
42 Complex a[N], b[N];
43 int ans[N];
44
45 void fft(Complex p[], int n, int type) {
46     for (int i = 0; i < n; i++) if (i < rev[i]) swap(p[i], p[rev[i]]);
47     for (int mid = 1; mid < n; mid <= 1) { //待合并区间的长度的一半
48         Complex wn(cos(pi / mid), type * sin(pi / mid)); //单位根
49         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
50             Complex w(1, 0); //幂
51             for (int k = 0; k < mid; k++, w = w * wn) { //枚举左半部分
52                 Complex x = p[j + k], y = w * p[j + mid + k]; //蝴蝶效应
53                 p[j + k] = x + y;
54                 p[j + mid + k] = x - y;
55             }
56         }
57     }
58 }
59
60 int main() {
61     scanf("%d", &n);
62     scanf("%s%s", x, y);
63     for (int i = n - 1; i >= 0; i--) a[n - 1 - i].real(x[i] - '0');
64     for (int i = n - 1; i >= 0; i--) b[n - 1 - i].real(y[i] - '0');
65     int top = 1, bit = 0;
66     while (top <= (n << 1)) top <= 1, bit++;
67     for (int i = 0; i < top; i++) rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
68     fft(a, top, 1); fft(b, top, 1);

```

```

69     for(int i=0;i<top;i++) a[i]=a[i]*b[i];
70     fft(a,top,-1);
71     for(int i=0;i<top;i++){
72         ans[i]+=(int)(a[i].real()/top+0.5);
73         if(ans[i]>=10){
74             ans[i+1]+=ans[i]/10; ans[i]%=10; top+=(i==top);
75         }
76     }
77     while(!ans[top]&&top>=1) top--;
78     top++;
79     while(--top>=0) printf("%d",ans[top]);
80 }

```

### 5.11 平面最近点对

```

1  int n;
2  int a[N],tot;
3
4  struct node{
5      double x,y;
6  }p[N];
7
8  bool cmp(node a,node b){
9      if(a.x!=b.x) return a.x<b.x;
10     return a.y<b.y;
11 }
12
13 bool cmp2(int i,int j){
14     return p[i].y<p[j].y;
15 }
16
17 double dist(int i,int j){
18     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));
19 }
20
21 double merge(int left,int right){
22     double d = 1e18;
23     if(left==right) return d;
24     if(left+1==right) return dist(left,right);
25     int mid=(left+right)>>1;
26     double d1=merge(left,mid),d2=merge(mid+1,right);
27     d=min(d1,d2);
28     tot=0;
29     for(int i=left;i<=right;i++) if(fabs(p[mid].x-p[i].x)<d) a[++tot]=i;
30     sort(a+1,a+1+tot,cmp2);
31     for(int i=1;i<=tot;i++){
32         for(int j=i+1;j<=tot&&p[a[j]].y-p[a[i]].y<d;j++){
33             double d3=dist(a[i],a[j]);
34             d=min(d,d3);
35         }
36     }
37     return d;
38 }
39
40 int main() {
41     #ifdef ACM_LOCAL
42         freopen("./std.in", "r", stdin);
43     #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

```

1  // 查找第一个相等的元素
2  int findFirstEqual(int[] array, int key) {
3      int left = 0;
4      int right = array.length - 1;
5      while (left <= right) {
6          int mid = (left + right) / 2;
7          if (array[mid] >= key)
8              right = mid - 1;
9          else
10             left = mid + 1;
11     }
12     if (left < array.length && array[left] == key)
13         return left;
14     return -1;
15 }
16
17 // 查找最后一个相等的元素
18 int findLastEqual(int[] array, int key) {
19     int left = 0;
20     int right = array.length - 1;
21     while (left <= right) {
22         int mid = (left + right) / 2;
23         if (array[mid] <= key)
24             left = mid + 1;
25         else
26             right = mid - 1;
27     }
28     if (right >= 0 && array[right] == key)
29         return right;
30     return -1;
31 }
32
33 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
34 double left=0,right=2000;
35 while(left+eps<=right){
36     double mid = (left+right)/2;
37     if(isOK(mid)) left=mid;
38     else right=mid;
39 }
40 printf("%lld\n",(ll)(1000*right));
41
42 // 查找最后一个等于或者小于key的元素
43 int findLastEqualSmaller(int[] array, int key) {
44     int left = 0;
45     int right = array.length - 1;
46     while (left <= right) {
47         int mid = (left + right) / 2;
48         if (array[mid] > key)
49             right = mid - 1;
50         else
51             left = mid + 1;
52     }
53     return right;
54 }
55

```

```

56 // 查找最后一个小于key的元素
57 int findLastSmaller(int[] array, int key) {
58     int left = 0;
59     int right = array.length - 1;
60     while (left <= right) {
61         int mid = (left + right) / 2;
62         if (array[mid] >= key)
63             right = mid - 1;
64         else
65             left = mid + 1;
66     }
67     return right;
68 }
69
70 // 查找第一个等于或者大于key的元素
71 int findFirstEqualLarger(int[] array, int key) {
72     int left = 0;
73     int right = array.length - 1;
74     while (left <= right) {
75         int mid = (left + right) / 2;
76         if (array[mid] >= key)
77             right = mid - 1;
78         else
79             left = mid + 1;
80     }
81     return left;
82 }
83
84 // 查找第一个大于key的元素
85 int findFirstLarger(int[] array, int key) {
86     int left = 0;
87     int right = array.length - 1;
88     while (left <= right) {
89         int mid = (left + right) / 2;
90         if (array[mid] > key)
91             right = mid - 1;
92         else
93             left = mid + 1;
94     }
95     return left;
96 }
97
98
99 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
100
101 //当存在极小值时
102 while(left+eps<=right){
103     double lm=left+(right-left)/3;
104     double rm=right-(right-left)/3;
105     if(calc(lm)>=calc(rm)) left=lm;
106     else right=rm;
107 }
108 printf("%.10lf\n",calc(left));
109
110 //当存在极大值时
111 while(left+eps<=right){
112     double lm=left+(right-left)/3;
113     double rm=right-(right-left)/3;

```

```

114     if(calc(lm)>=calc(rm)) right=rm;
115     else left=lm;
116 }
117 printf("%.10lf\n",calc(left));

```

## 6.2 离散化

```

1 1、离散化
2 for (int i = 1; i <= n; i++) {
3     scanf("%d", &a[i]);
4     b[i] = a[i];
5 }
6 sort(b + 1, b + 1 + n);
7 m = unique(b + 1, b + 1 + n) - b - 1;
8 for (int i = 1; i <= n; i++) {
9     a[i] = lower_bound(b + 1, b + 1 + m, a[i]) - b;
10 }

```

## 6.3 斯坦纳树

1 给定 $n$ 个点, $m$ 条边,请选择一些边,使得 $1 \leq i \leq d(1 \leq d \leq 4)$ , $i$ 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。

2 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)

3 有两种转移方法:

4 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

5 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

6 对于第一种转移, 我们直接枚举子集

7 对于第二种转移, 我们用spfa进行状态转移

```

8
9 int n,m,d;
10
11 struct node{
12     int to,v,nx;
13 }edge[N<<1];
14 int tot,head[N];
15
16 void add(int from,int to,int v) {
17     edge[tot].to=to;
18     edge[tot].v=v;
19     edge[tot].nx=head[from];
20     head[from]=tot++;
21 }
22
23 class SteinerTree{
24 public:
25     int f[260][N],g[260];
26     queue<int>q;
27     bool in[N];
28
29     void spfa(int S) {
30         while (!q.empty()) {
31             int from=q.front(); q.pop();
32             in[from]=0;
33             for (int i=head[from];i;i=edge[i].nx) {
34                 int to=edge[i].to;
35                 if (f[S][to]>f[S][from]+edge[i].v) {
36                     f[S][to]=f[S][from]+edge[i].v;

```

```

37         if (!in[to]) {
38             in[to]=1;
39             q.push(to);
40         }
41     }
42 }
43 }
44 }
45
46 void solve(){
47     memset(f,0x3f,sizeof(f));
48     int cnt=2*d;
49     for (int i=1;i<=d;i++) f[1<<i-1][i]=0,f[1<<d+i-1][n-i+1]=0;
50     int top=(1<<cnt);
51     for (int S=1;S<top;S++) {
52         for (int s=(S-1)&S;s=s=(s-1)&S) {
53             int t=S^s;
54             for (int i=1;i<=n;i++)
55                 f[S][i]=min(f[S][i],f[s][i]+f[t][i]);
56         }
57         for (int i=1;i<=n;i++)
58             if (f[S][i]<INF&&!in[i])
59                 q.push(i),in[i]=1;
60         spfa(S);
61     }
62     memset(g,0x3f,sizeof(g));
63     top=(1<<d);
64     for (int S=1;S<top;S++)
65         for (int i=1;i<=n;i++)
66             g[S]=min(g[S],f[S^(S<<d)][i]);
67     for (int S=1;S<top;S++)
68         for (int s=(S-1)&S;s=s=(s-1)&S)
69             g[S]=min(g[S],g[s]+g[S^s]);
70     printf("%d",g[top-1]==INF? -1:g[top-1]);
71 }
72 }steinerTree;
73
74 int main(){
75     int cin_x,cin_y,cin_v;
76     scanf("%d%d%d",&n,&m,&d);
77     tot=1;
78     for(int i=1;i<=m;i++){
79         scanf("%d%d%d",&cin_x,&cin_y,&cin_v);
80         add(cin_x,cin_y,cin_v);
81         add(cin_y,cin_x,cin_v);
82     }
83     steinerTree.solve();
84 }

```

## 6.4 子矩阵问题

```

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

```

```

8     for (int j = 1; j <= m; j++) h[i][j] = (s[i][j] == '1') ? h[i - 1][j] + 1 : 0;
9     h[i][m + 1] = 0; st[top = 0] = 0;
10    for (int j = 1; j <= m + 1; j++) {
11        if (h[i][j] == h[i][st[top]]) st[top] = j;
12        else if (h[i][j] > h[i][st[top]]) st[++top] = j;
13        else {
14            int t = st[top];
15            do {
16                hi = h[i][st[top]]; li = (t - st[--top]); //求长度和高度
17                ans.push_back(hi * li); //计算面积
18                ans.push_back(max((hi - 1) * li, hi * (li - 1)));
19            } while (h[i][j] < h[i][st[top]]);
20            if (h[i][j] > h[i][st[top]]) top++;
21            st[top] = j;
22        }
23    }
24    sort(ans.begin(), ans.end(), greater<int>());
25    ans.resize(2);
26 }
27 printf("%d\n", ans[1]);
28 }

```

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

```

31 void solve(){
32     int maxl,maxr;
33     for(int i=1;i<=m;i++) r[0][i]=m;
34     for(int i=1;i<=n;i++){
35         maxl=1; maxr=m;
36         for(int j=1;j<=m;j++){
37             if(s[i][j]=='0'){
38                 maxl=j+1;
39                 h[i][j]=l[i][j]=0;
40             }else{
41                 h[i][j]=h[i-1][j]+1;
42                 l[i][j]=max(maxl,l[i-1][j]);
43             }
44         }
45         for(int j=m;j>=1;j--){
46             if(s[i][j]=='0'){
47                 maxr=j-1;
48                 r[i][j]=m;
49             }else{
50                 r[i][j]=min(maxr,r[i-1][j]);
51                 int R=r[i][j]-l[i][j]+1,H=h[i][j];
52                 ans=max(ans,R*H);
53             }
54         }
55     }
56     printf("%d\n",ans);
57 }

```

58 //简单悬线法

```

60 for (int i = 1; i <= n; i++)
61     for (int j = 1; j <= m; j++)
62         scanf(" %c", s[i] + j), lft[i][j] = rgt[i][j] = j, up[i][j] = 1;
63 for (int i = 1; i <= n; i++) {
64     for (int j = 1; j <= m; j++)
65         if (s[i][j] == '0' && s[i][j - 1] == '0') lft[i][j] = lft[i][j - 1];

```



```

66     for (int j = m; j >= 1; j--)
67         if (s[i][j] == '0' && s[i][j + 1] == '0') rgt[i][j] = rgt[i][j + 1];
68     }
69     for (int i = 1; i <= n; i++)
70         for (int j = 1; j <= m; j++) {
71             if (i > 1 && s[i][j] == '0' && s[i - 1][j] == '0') {
72                 lft[i][j] = max(lft[i][j], lft[i - 1][j]);
73                 rgt[i][j] = min(rgt[i][j], rgt[i - 1][j]);
74                 up[i][j] = up[i - 1][j] + 1;
75             }
76             ans = max(ans, (rgt[i][j] - lft[i][j] + 1) * up[i][j]);
77         }
78
79 3、优先队列+并查集求第二大1矩阵
80 int find(int x){
81     if (par[x] == x) return x;
82     return par[x] = find(par[x]);
83 }
84
85 void merge(int x, int y){
86     int fx=find(x),fy=find(y);
87     par[fy]=fx;
88     len[fx]+=len[fy];
89     return;
90 }
91
92 int main() {
93     scanf("%d%d",&n,&m);
94     int ans=0,ans2=0;
95     for(int i=1;i<=n;i++){
96         scanf("%s",s+1);
97         for(int j=1;j<=m;j++){
98             if(s[j]=='0') h[j]=0;
99             else{
100                 h[j]++;
101                 q.push(node(j,h[j]));
102             }
103             len[j]=1;par[j]=j;vis[j]=0;
104         }
105         while(!q.empty()){
106             node t=q.top(); q.pop();
107             int pos=t.pos,hi=t.height;
108             vis[pos]=1;
109             if(vis[pos-1]) merge(pos-1,pos);
110             if(vis[pos+1]) merge(pos+1,pos);
111             int f=find(pos);
112             if(len[f]*hi>=ans) ans2=ans,ans=len[f]*hi;
113             else ans2=max(ans2,len[f]*hi);
114             ans2=max(ans2,max((len[f]-1)*hi,len[f]*(hi-1)));
115         }
116     }
117     printf("%d\n",ans2);
118 }
119
120 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
121 void solve(){
122     int ans=0,ans2=0;
123     for(int i=1;i<=n;i++){
124         for(int j=1;j<=m;j++) { mx[j]=-inf; mn[j]=inf;}

```

```

125     for(int j=i;j<=n;j++){
126         for(int k=1;k<=m;k++) mx[k]=max(mx[k],a[j][k]);
127         for(int k=1;k<=m;k++) mn[k]=min(mn[k],a[j][k]);
128         int l=1,head=1,tail=0,head2=1,tail2=0;
129         for(int r=1;r<=m;r++){
130             while(head<=tail&&mx[r]>=mx[q[0][tail]]) tail--;
131             while(head2<=tail2&&mn[r]<=mn[q[1][tail2]]) tail2--;
132             q[0][++tail]=r;
133             q[1][++tail2]=r;
134             while(l<=r&&mx[q[0][head]]-mn[q[1][head2]]>k){
135                 l++;
136                 if(q[0][head]<l) head++;
137                 if(q[1][head2]<l) head2++;
138             }
139             ans=max(ans,(j-i+1)*(r-l+1)); //求最大子矩阵
140             ans+=r-l+1; //求所有满足条件的子矩阵
141         }
142     }
143 }
144 printf("%d\n",ans);
145 }

```

## 6.5 矩阵模板

```

1  1、
2  friend bool operator< (node a,node b){
3      return a.step>b.step;
4  }
5  优先队列将步数较小的放在前面
6
7  2、
8  do {
9      if (check()) {
10         .....
11     }
12 } while (next_permutation(a + 1, a + 1 + n));
13 全排列函数，注意最初需要将a从小到大进行排序。
14
15 矩阵模板：
16 mat Mul(mat a, mat b) {
17     mat c;
18     for (int i = 0; i < ac.tot; i++) {
19         for (int j = 0; j < ac.tot; j++) {
20             ull sum = 0;
21             for (int k = 0; k < ac.tot; k++)
22                 sum = sum + a.arr[i][k] * b.arr[k][j];
23             c.arr[i][j] = sum;
24         }
25     }
26 }
27 return c;
28 }
29
30 mat quick(mat a, int b) {
31     mat res = ones;
32     while (b) {
33         if (b & 1)
34             res = Mul(res, a);

```

```

35         b >>= 1;
36         a = Mul(a, a);
37     }
38     return res;
39 }
40
41 mat add(mat a, mat b) {
42     mat c;
43     for (int i = 0; i < ac.tot; i++) {
44         for (int j = 0; j < ac.tot; j++) {
45             c.arr[i][j] = a.arr[i][j] + b.arr[i][j];
46         }
47     }
48     return c;
49 }
50
51 求:  $A^1 + A^2 + A^3 + A^4 \dots + A^K$ 
52 mat getsum(int k) {
53     if (k == 1)
54         return A;
55     mat t = getsum(k / 2);
56     if (k & 1) {
57         mat reminder = quick(A, k);
58         mat cur = quick(A, (k - 1) / 2);
59         t = add(t, Mul(cur, t));
60         t = add(reminder, t);
61     }
62     else {
63         mat cur = quick(A, k / 2);
64         t = add(t, Mul(cur, t));
65     }
66     return t;
67 }

```

## 6.6 汉诺塔

```

1  汉诺塔:
2      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
3  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,
4  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。
5      □ 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
6  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。
7      □ 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
8  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。
9      □ 反复进行□ □ 操作, 最后就能按规定完成汉诺塔的移动。
10     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:
11     如3阶汉诺塔的移动: A→C, A→B, C→B, A→C, B→A, B→C, A→C
12
13 void Move(int n, char a, char b){
14     printf("第%d次移动 Move %d: Move from %c to %c !\n", ++cnt, n, a+'A', b+'A');
15 }
16
17 void Hanoi(int n, int a, int b, int c){

```

```

18     if(n==1){
19         Move(n,a,c);
20     }else{
21         Hanoi(n - 1, a, c, b);
22         Move(n, a, c);
23         Hanoi(n - 1, b, a, c);
24     }
25 }
26
27
28 求汉诺塔具体操作数结果:
29 int main(){
30     int ab=0,ac=1,ba=2,bc=3,ca=4,cb=5;
31     scanf("%d",&n);
32     ret[ac]++; dp[1][ac]=1;
33     for(int i=2;i<=n;i++){
34         if(i%2==0){
35             dp[i][ab]+=dp[i-1][ac]+dp[i-1][cb];
36             dp[i][bc]+=dp[i-1][ac]+dp[i-1][ba];
37             dp[i][ca]+=dp[i-1][ba]+dp[i-1][cb];
38         }else{
39             dp[i][ac]+=dp[i-1][ab]+dp[i-1][bc];
40             dp[i][ba]+=dp[i-1][bc]+dp[i-1][ca];
41             dp[i][cb]+=dp[i-1][ab]+dp[i-1][ca];
42         }
43         for(int j=0;j<6;j++) ret[j]+=dp[i][j];
44     }
45     printf("A->B:%lld\n", ret[0]);
46     printf("A->C:%lld\n", ret[1]);
47     printf("B->A:%lld\n", ret[2]);
48     printf("B->C:%lld\n", ret[3]);
49     printf("C->A:%lld\n", ret[4]);
50     printf("C->B:%lld\n", ret[5]);
51     printf("SUM:%lld\n", pw(n)-1);
52 }

```

## 6.7 前缀和与差分

```

1 1、前缀和与差分
2     对于一个数组a定义数组s[i]=sigma(j=1,i)a[j]
3 //为了避免数组越位,下标从1开始用
4 for(int i=1;i<=n;i++)
5     s[i]=s[i-1]+a[i];
6
7     定义数组d[i]= 1、i==0,di[i]=a[i] 2、i>=1,d[i]=a[i]-a[i-1]
8
9 //为了避免数组越位,下标从1开始用
10 for(int i=n;i>=1;i--)
11     d[i]=a[i]-a[i-1];
12
13     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
14 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]....
15 a[n]全部加上x。
16 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。
17 void add(int l,int r,int x){

```

```

18     d[l]+=x;
19     d[r+1]-=x;
20 }
21
22 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。
23
24 2、静态维护区间加等差数列的求和问题
25     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
    个首项为a，公差为k的等差数列。
26 其中[l,r]区间分别加上[a,a+k,a+2k,...a+(r-l)k]
27
28 int n,m,d2[N],l,r,a,k;
29 void add(int l,int r,int a,int k){
30     d2[l]+=a;
31     d2[l+1]+=k-a;
32     d2[r+1]-=(r-l+1)*k+a;
33     d2[r+2]-=(l-r)*k-a;
34 }
35 void iter(){
36     for(int i=1;i<=n;++i) d2[i]+=d2[i-1];
37 }
38
39 int main(){
40     scanf("%d%d",&n,&m);
41     for(int i=1;i<=m;++i){
42         scanf("%d%d%d%d",&l,&r,&a,&k);
43         add(l,r,a,k);
44     }
45     iter(); //第一次为了将每项额外加的补齐
46     iter(); //还原完整数组
47     for(int i=1;i<=n;++i) printf("%d%c",d2[i],i==n?'\\n':' ');
48     return 0;
49 }
50
51 3、二维前缀和与差分
52     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
53 //为了避免数组越位，下标从1开始
54 for(int i=1;i<=n;i++){
55     for(int j=1;j<=m;j++){
56         s[i][j]=s[i-1][j]+s[i][j-1]-s[i-1][j-1]+a[i][j];
57     }
58 }
59     那么定义d[i][j]为差分数组
60 为了避免数组越位，下标从1开始
61 for(int i=n;i;i--){
62     for(int j=m;j;j--){
63         d[i][j]=a[i][j]-a[i-1][j]-a[i][j-1]+a[i-1][j-1];
64     }
65 }
66
67 静态数组的求和问题
68 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
69 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]
70 部分被减了两次，所以再加上一次。
71 ll sum(int l1,int r1,int l2,int r2){
72     return s[r1][r2]-s[l1-1][r2]-s[r1][l2-1]+s[l1-1][l2-1];

```

```

73 }
74
75 进行m次区间修改后的静态单点求值问题
76 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）
77 void add(int l1,int r1,int l2,int r2,int x){
78     d[l1][l2]+=x;
79     d[r1+1][l2]-=x;
80     d[l1][r2+1]-=x;
81     d[r1+1][r2+1]+=x;
82 }
83 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。
84
85
86 4、高维前缀和
87 同理三维只需要三个维度进行处理即可
88 for(int i=1;i<=n;++i){
89     for(int j=1;j<=m;++j){
90         a[i][j]=a[i][j]+a[i][j-1];
91     }
92 }
93 for(int i=1;i<=n;++i){
94     for(int j=1;j<=m;++j){
95         a[i][j]=a[i][j]+a[i-1][j];
96     }
97 }
98
99 5、状压dp前缀和
100 for(int i=0;i<w;++i)//依次枚举每个维度
101 {
102     for(int j=0;j<(1<<w);++j)//求每个维度的前缀和
103     {
104         if(j&(1<<i))s[j]+=s[j^(1<<i)];
105     }
106 }
107
108
109 6、菱形差分
110 题目：
111 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，
112 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)
113 (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
114
115 输入：
116 第一行三个整数n, m, q分别表示矩形的长/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
117 保证l为奇数！
118
119 4 5 4
120 1 2 2 1
121 1 3 3 5
122 1 3 2 3
123 2 2 4 3
124
125 轰炸后结果为：
126

```

```

127 0 0 1 1 0
128 0 3 2 2 1
129 2 2 2 1 1
130 0 2 1 1 0
131 最后把所有元素异或后为2
132
133 代码:
134 int n,m,q,base=500;
135 int op,x,y,L;
136 int a[N][N],b[N][N];
137
138 void down(int x,int y,int L){
139     a[x+L+1][y]++;
140     a[x+1][y+L]--;
141
142     b[x+1][y-L+1]++;
143     b[x+L+1][y+1]--;
144 }
145
146 void up(int x,int y,int L){
147     a[x-L][y]++;
148     a[x+1][y-L-1]--;
149
150     b[x-L][y+1]--;
151     b[x+1][y+2+L]++;
152 }
153
154 int main() {
155     read(n); read(m); read(q);
156     while(q--){
157         read(op); read(x); read(y); read(L);
158         x+=base; y+=base;
159         if(op==1) down(x,y,L/2);
160         up(x,y,L/2);
161     }
162     int ans=0;
163     for(int i=1;i<=n+base*2;i++){
164         int x=0;
165         for(int j=1;j<=m+base*2;j++){
166             x+=a[i][j]+b[i][j];
167             if(i>base&&i<=n+base&&j>base&&j<=m+base) ans^=x;
168             a[i+1][j-1]+=a[i][j];
169             b[i+1][j+1]+=b[i][j];
170         }
171     }
172     printf("%d\n",ans);
173 }

```

## 6.8 stl

- 1 1、lower\_bound,upper\_bound
- 2 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。
- 3
- 4 在从小到大的排序数组中:
- 5 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。
- 6

```

7      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
8 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。
9
10     在从大到小的排序数组中，重载lower_bound()和upper_bound():
11     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
12 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。
13     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，
14 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。
15
16
17 //数组测试
18 int a[N];
19
20 int main(){
21     int n=10;
22     //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
23     for(int i=0;i<n;i++) a[i]=i;
24     int i=lower_bound(a,a+n,n+n)-a;
25     printf("%d %d\n",i,a[i]);
26     //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
27     i=upper_bound(a,a+n,100)-a;
28     printf("%d %d\n",i,a[i]);
29     //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
30     for(int i=0;i<n;i++) a[i]=n-i-1;
31     i=lower_bound(a,a+n,-1,greater<int>())-a;
32     printf("%d %d\n",i,a[i]);
33     //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
34     i=upper_bound(a,a+n,-999,greater<int>())-a;
35     printf("%d %d\n",i,a[i]);
36
37     //情况上面情况一样
38     vector<int>v;
39     for(int i=0;i<n;i++) v.push_back(i);
40     int i=lower_bound(v.begin(),v.end(),23)-v.begin();
41     printf("%d %d\n",i,v[i]);
42 }
43
44
45 2、结构体中比较函数作用于数组和set
46 //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
47 struct node{
48     int x,y;
49     bool operator <(const node&t)const{
50         if(x==t.x) return y>t.y;
51         return x<t.x;
52     }
53 };
54
55 int main(){
56 #ifdef ACM_LOCAL
57     freopen("./std.in", "r", stdin);
58     //freopen("./std.out", "w", stdout);
59 #endif
60     set<node>st;
61     st.insert(node{1,1});

```



```

62     st.insert(node{1,3});
63     st.insert(node{2,2});
64     node p[N];
65     p[1]=node{1,1}; p[2]=node{1,3}; p[3]=node{2,2};
66     sort(p+1,p+1+3);
67     //(1,3)(1,1)(2,2)
68 }
69
70 3、优先队列的比较函数
71 //跟上述情况相反
72 struct node{
73     int x,y;
74     bool operator <(const node&t)const{
75         if(x==t.x) return y>t.y;
76         return x<t.x;
77     }
78 };
79
80 int main(){
81 #ifdef ACM_LOCAL
82     freopen("./std.in", "r", stdin);
83     //freopen("./std.out", "w", stdout);
84 #endif
85     priority_queue<node>q;
86     q.push(node{1,3});
87     q.push(node{1,1});
88     q.push(node{2,1});
89     while(!q.empty()){
90         node t=q.top(); q.pop();
91         printf("%d %d\n",t.x,t.y);
92     }
93     //(2,1)(1,1)(1,3)
94     set<int>st;
95     for(int i=1;i<=5;i++) st.insert(i);
96     cout<<* (--st.end())<<endl;
97     //5
98 }

```

## 6.9 最长上升子序列

```

1  问题1:
2      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
3      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
4
5  int main(){
6      int Case=1;
7      while(scanf("%d",&n)==1){
8          for(int i=1;i<=n;i++) scanf("%d%d",&a[i].x,&a[i].y);
9          sort(a+1,a+1+n); //x<t.x
10         set<int>st;
11         st.insert(a[1].y);
12         for(int i=2;i<=n;i++){
13             auto it=st.lower_bound(a[i].y);
14             if(it==st.end()){
15                 st.insert(a[i].y);
16             }else{
17                 st.erase(it);
18                 st.insert(a[i].y);

```

```

19         }
20     }
21     printf("Case %d:\n",Case++);
22     if(st.size()==1){
23         printf("My king, at most %d road can be built.\n\n",st.size());
24     }else{
25         printf("My king, at most %d roads can be built.\n\n",st.size());
26     }
27 }
28 return 0;
29 }

```

30 问题2:

31 现在你有N块矩形木板，第i块木板的尺寸是 $X_i * Y_i$ ，第i块木板能放在第j块木板上方当且仅当 $X_i < X_j$ 且 $Y_i < Y_j$ ，  
 32 于是你很可能没法  
 33 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。

34 根据Dilworth定理，最小组数等于 $Z_i$ 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

```

37
38 int n;
39 struct node{
40     int x,y,id;
41 }p[N];
42
43 bool cmp(node s,node t){
44     if(s.x==t.x) return s.y>t.y;
45     return s.x>t.x;
46 }
47
48 int bcc[N];
49 set<pii>st;
50
51 int main(){
52 #ifdef ACM_LOCAL
53     freopen("./std.in", "r", stdin);
54     //freopen("./std.out", "w", stdout);
55 #endif
56     scanf("%d",&n);
57     for(int i=1;i<=n;i++){
58         scanf("%d%d",&p[i].x,&p[i].y);
59         p[i].id=i;
60     }
61     sort(p+1,p+1+n,cmp);
62     int tot=1;
63     for(int i=1;i<=n;i++){
64         auto it=st.lower_bound(make_pair(p[i].y,0));
65         if(it==st.end()){
66             bcc[p[i].id]=tot;
67             st.insert(make_pair(p[i].y,tot));
68             tot++;
69         }else{
70             bcc[p[i].id]=(*it).second;
71             st.erase(it);
72             st.insert(make_pair(p[i].y,bcc[p[i].id]));
73         }
74     }

```

```
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

1 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。

2 其基础代码为:

```
3 for(int len=2;len<=n;len++){
4     for(int i=1;i+len-1<=n;i++){
5         int j = i+len-1;
6         for(int k=i;k<j;k++){
7             dp[i][j]=max(dp[i][j],dp[i][k]+dp[k+1][j]+cost[i]);
8         }
9     }
10 }
```

11 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

12 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。

13

14 下面对一些精彩的题目进行分析:

15

16 1) Cake(zoj 3537)

17 题意:

18 有一块多边形蛋糕,切一刀的代价为 $|x_1+x_2|*|y_1+y_2|*p$ ,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。

19

20 思路:

21 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 $n \rightarrow 2n$ ,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

22 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。

23

24 2) Coloring Brackets(<https://codeforces.com/problemset/problem/149/D>)

25 题意:

26 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

27 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)。

28

29 思路:

30 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

31 最大数量。那么接下来只需要分类讨论即可获得最优解。

32

33 代码:

```
34 for(int L=3;L<=len;L++){
35     for(int i=1;i+L-1<=len;i++){
36         int j = i+L-1;
37         if(s[i]=='(' && to[i]==j){
38             dp[i][j][0][1] = (dp[i][j][0][1]+dp[i+1][j-1][1][0]+dp[i+1][j-1][0][2]
39             +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
40             -1][1][2])%mod;
41             dp[i][j][1][0] = (dp[i][j][1][0]+dp[i+1][j-1][0][1]+dp[i+1][j-1][2][0]
42             +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
43             -1][2][1])%mod;
44             dp[i][j][2][0] = (dp[i][j][2][0]+dp[i+1][j-1][0][1]+dp[i+1][j-1][1][0]
45             +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
46             -1][2][1])%mod;
```

```

46         }else if(s[i]=='(' && s[j]==')'){
47             int k = to[i];
48             for(int x=0;x<3;x++){
49                 for(int y=0;y<3;y++){
50                     for(int q1=0;q1<3;q1++){
51                         for(int q2=0;q2<3;q2++){
52                             if(q1==q2 && q1!=0)
53                                 continue;
54                             dp[i][j][x][y]=(dp[i][j][x][y]+dp[i][k][x][q1]*dp[k+1][
j][q2][y])%mod;
55                         }
56                     }
57                 }
58             }
59         }
60     }
61 }

```

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。

64  
65 3) You Are the One(hdu 4283)

66 题意:

67 有一个序列，每个人带着屌丝值 $d_i$ ，第 $k$ 个上场的人会增加评委 $(k-1)*d_i$ ，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

68 总和。

69

70 思路:

71 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 $k$ 个人进入，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。

72

73

74 反思:

75 赋予 $k$ 特殊含义，来求解。

76

77 4) Palindrome subsequence(hdu 4632)

78 题意:

79 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。

80

81 反思:

82 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  $if(s[i-1]==s[j-1]) dp[i][j]=(dp[i][j]+1+dp[i+1][j-1])\%mod;$

83

84

85 5) Two Rabbits(hdu 4745)

86 题意:

87 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。

88

89 思路:

90 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。

91

92 代码:

```

93 for(int i=1;i<=n;i++){
94     dp[i][i]=1;
95     for(int len = 2;len<=n;len++){
96         for(int i = 1;i+len-1<=2*n;i++){
97             int j = i+len-1;

```

```

98     dp[i][j]=max(dp[i+1][j],dp[i][j-1]);
99     if(a[i]==a[j])
100         dp[i][j]=max(dp[i][j],dp[i+1][j-1]+2);
101 }
102 }
103 int res=1;
104 for(int i=1;i<=n;i++){
105     int j = i+n-1;
106     for(int k=i;k<j;k++){
107         res = max(res,dp[i][k]+dp[k+1][j]);
108     }
109 }
110 printf("%d\n",res);
111

```

112 6) Sit sit sit (hdu 5151)

113 题意:

114 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

115 现求所有可行方案mod(1e9+7)

116

117 思路:

118 设第k个人为最后上座，加上一些组合数知识即可写出。

119

120 7) D-game

121 题意:

122 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( $X \geq 2$ )个连续数字；2. 检查1选择

123 的X个数字是否构成等差数列，且公差  $d \in D$ ；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

124 删除几个数。

125

126 代码:

```

127 for(int len=2;len<=n;len++){
128     for(int i=1;i+len-1<=n;i++){
129         int j = i+len-1;
130         if(len==2){
131             if(mp[a[j]-a[i]])
132                 dp[i][j]=1;
133         }else if(len==3){
134             if(mp[a[j-1]-a[i]]&&2*a[j-1]==a[j]+a[i])
135                 dp[i][j]=1;
136         }else{
137             if(mp[a[j]-a[i]]&&dp[i+1][j-1])
138                 dp[i][j]=1;
139             for(int k=i+1;k<j;k++){
140                 dp[i][j] |= dp[i][k]&dp[k+1][j];
141             }
142         }
143     }
144 }
145 memset(res,0, sizeof(res));
146 for(int j=1;j<=n;j++){
147     res[j]=res[j-1];
148     for(int i=1;i<=j;i++){
149         if(dp[i][j])
150             res[j]=max(res[j],res[i-1]+j-i+1);
151     }
152 }
153 printf("%d\n",res[n]);

```

## 7.2 一般 dp

1 1.Photo Processing(cf 883I)

2 题意:

3 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

4

5 思路:

6 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

7

8 代码:

9 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 $dp[i]=1$

```
10 bool check(int differ){
11     memset(dp,0, sizeof(bool)*(n+2));
12     int index=1; dp[0]=1;
13     for(int i=1; i<=n; i++){
14         while(a[i]-a[index]>differ)
15             index++;
16         while(i-index+1>=k){
17             if(dp[index-1]){
18                 dp[i]=1; break;
19             }
20             index++;
21         }
22     }
23     return dp[n];
24 }
```

25 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

```
26 bool check(int differ){
27     int last=0;
28     for(int i=k; i<=n; i++){
29         int j = dp[i-k];
30         if(a[i]-a[j+1]<=differ)
31             last=i;
32         dp[i]=last;
33     }
34     return dp[n]==n;
35 }
```

36

37 2. 小D的剧场(<https://ac.nowcoder.com/acm/contest/369/A>)

38 题意:

39 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。

40

41 思路:

42 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

43 第二个位置为j, 第三个位置为k的方案数, 可列出:  $dp[t][i][j] = (dp[t][i][j] + dp[t-1][j][k]) \% mod$ ;

44

45 3、最少拦截系统(hdu1257)

46 题意:

47 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

48 将导弹全部射下。

49

50 思路:

51 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

52 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

53 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。

54

### 55 3、Max Sum Plus Plus(hdu1024)

56 题意：

57 给你a[N]数组，( $1 \leq n \leq 1e6$ )，求k个区间段最大值为多少。

58 思路：

59 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：

60  $dp[i][j] = \max\{dp[i][j-1], \max\{dp[i-1][t] \mid (i-1 \leq t \leq j-1)\} + a[j]\}$ 。前者因为dp[i][j-1]的值已知，若不增加i，只需要

61 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，

62 且( $1 \leq n \leq 1e6$ )，因此我们可以设 $w[i][j] = \max\{dp[i][t] \mid (i \leq t \leq j)\} = \max\{w[i][j-1], dp[i][j]\}$ ，从而我们可以将方程化为：

63 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

64 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)  $w[t^1][j] = \max\{dp[j], w[t^1][j-1]\}$

65 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，

66 只需要求解n-(m-i)个数即可

67

68 核心代码：

69 `scanf("%d",&n);`

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

71 `scanf("%d",&a[i]);`

72 `pre[i]=pre[i-1]+a[i];`

73 `w[0][i]=0;`

74 `}`

75 `int t=1;`

76 `for(int i=1;i<=k;i++){`

77 `dp[i]=w[t][i]=pre[i];`

78 `for(int j=i+1;j<=n+i-k;j++){`

79 `dp[j]=max(dp[j-1],w[t^1][j-1])+a[j];`

80 `w[t][j]=max(dp[j],w[t][j-1]);`

81 `}`

82 `t^=1;`

83 `}`

84 `printf("%lld\n",w[t^1][n]);`

85

### 86 3、Phalanx(hdu2859)

87 题意：

88 给你一个n\*n的矩阵，( $1 \leq n \leq 1000$ )，求该矩阵中最大的对称矩阵，对称先为左下角到右上角。

89

90 思路：

91 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

92 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

93 点即可。

94

95 反思：

96 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。

97

### 98 4、Making the Grade(poj3666)

99 题意：

100 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。

101



102 思路:

103 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

104  $dp[i][j]$ 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 $dp[i][j] = \min(dp[i-1][k] (1 \leq k \leq j)) + \text{abs}(a[i] - b[j])$ 。

105 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。

106

107

108 反思:

109 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。

110

111 5、Jury Compromise(poj1015)

112 题意:

113 有评审员会打出两个值,  $pi$ 和 $di$  ( $0 \leq pi, di \leq 20$ )。现在有 $n$  ( $1 \leq n \leq 200$ )个评审员, 要选取 $m$  ( $1 \leq m \leq 20$ ), 其中要满足 $Pi = \sum(pi)$ ,

114  $Di = \sum(di)$ , 要求 $|Pi - Di|$ 最小, 如有相同则 $Pi + Di$ 越大越好, 并输出方案。

115

116 思路:

117 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,

118 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。

119 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 $state[i][j]$ :

120 `typedef struct {`

121 `int sum;`

122 `bool vaild;`

123 `int path[M];`

124 `}node;`

125  $state[i][j]$ 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。

126 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。

127

128 核心代码:

129 `int dp() {`

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

131 `int sub = a[i][0] - a[i][1];`

132 `int add = a[i][0] + a[i][1];`

133 `for (int j = m; j >= 1; j--) {`

134 `for (int x = 0; x <= 800; x++) {`

135 `if (state[j - 1][x].vaild && x + sub >= 0 && x + sub <= 800) {`

136 `if (state[j - 1][x].sum + add >= state[j][x + sub].sum) {`

137 `state[j][x + sub].sum = state[j - 1][x].sum + add;`

138 `state[j][x + sub].vaild = 1;`

139 `for (int index = 1; index < j; index++)`

140 `state[j][x + sub].path[index] = state[j - 1][x].path[index];`

141 `};`

142 `state[j][x + sub].path[j] = i;`

143 `}`

144 `}`

145 `}`

146 `}`

147 `for (int i = 0; i <= Base; i++) {`

148 `if (state[m][i + Base].vaild || state[m][Base - i].vaild) {`

149 `int sumMax = -1;`

150 `if (state[m][i + Base].vaild)`

151 `sumMax = state[m][i + Base].sum;`

```

152         if (state[m][Base - i].vaild&&state[m][Base - i].sum > sumMax)
153             return Base - i;
154         return Base + i;
155     }
156 }
157 }
158

```

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( $1 \leq n, m \leq 100$ ), 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

```

168 dp[0][0][0][0]=1;
169 for(int i=1, p = 1; i <= n; i++, p ^= 1){
170     for (int j = 0; j <= i; j++)
171         for (int k = 0; k <= j; k++)
172             for (int q = 0; q <= k; q++)
173                 dp[p][j][k][q] = 0;
174     for(int j=0; j<i; j++) {
175         for (int k = 0; k <= j; k++) {
176             for (int q = 0; q <= k; q++) {
177                 Mod(dp[p][j][k][q] += dp[p^1][j][k][q]);
178                 Mod(dp[p][i-1][k][q] += dp[p^1][j][k][q]);
179                 Mod(dp[p][i-1][j][q] += dp[p^1][j][k][q]);
180                 Mod(dp[p][i-1][j][k] += dp[p^1][j][k][q]);
181             }
182         }
183     }
184     for(int j=0; j<i; j++){
185         for(int k=0; k<=j; k++){
186             for(int q=0; q<=k; q++){
187                 for(pii t:v[i]){
188                     if(1+(j>=t.first)+(k>=t.first)+(q>=t.first)!=t.second)
189                         dp[p][j][k][q]=0;
190                 }
191             }
192         }
193     }
194 }
195 ll ans=0;
196 for(int i=0, p=n&1; i<n; i++){
197     for(int j=0; j<=i; j++){
198         for(int k=0; k<=j; k++){
199             Mod(ans += dp[p][i][j][k]);
200         }
201     }
202 }
203

```

## 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 $a_i$ 的寒冷值侵袭, 她可以选择在任何一些

207 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！

208 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

209  $1 \leq n \leq 5000, 0 \leq k \leq 1e15, 0 \leq a \leq 1e9$

210

211 思路：

212 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 $i \sim n$ 时间内，最多可以抖动j次的最小时间，那么转移方程为

213  $dp[i][j] = \min(dp[i+1][j] + a[i], dp[i+1][j-1])$ 。然后从 $1 \rightarrow n$ 遍历，设sum为到i位置时寒冷总和，那么存在如下关系：

214 1、 $sum + a[i] + dp[i+1][cnt] \leq k$ ，则 $sum += a[i]$  2.  $sum + a[i] + dp[i+1][cnt] > k$ ，则 $cnt--$

### 7.3 数位 dp

1 1、不要62(hdu2089)

2 题意：

3 求出区间 $[n, m]$ 中数位没有62和4的个数

4 思路：

5 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。

6 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：

7 1)、 $dp[i][j] = 0$  ( $j=4$ ) 2)、 $dp[i][j] = \sum dp[i-1][j']$  ( $j'=0, 1, \dots, 9$ )，当 $j=6$ 时，需要减去 $dp[i-1][2]$

8 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。

9 最终即可获得答案。

10

11 代码：

12 1、

```

13 void init(){
14     dp[0][0]=1;
15     for(int i=1; i<10; i++){
16         for(int j=0; j<10; j++){
17             if(j==4){
18                 dp[i][j]=0;
19             }else if(j!=6){
20                 for(int k=0; k<=9; k++){
21                     dp[i][j] += dp[i-1][k];
22                 }else if(j==6){
23                     for(int k=0; k<=9; k++){
24                         dp[i][j] += dp[i-1][k];
25                         dp[i][j] -= dp[i-1][2];
26                     }
27                 }
28             }
29         }
30     }
31     int v[N+10];
32     ll solve(int di){
33         ll res=0;
34         v[0]=0;
35         while(di){
36             v[++v[0]] = di%10;
37             di /= 10;
38         }
39         v[v[0]+1] = 0;
40         for(int i=v[0]; i>=1; i--){
41             for(int j=0; j<v[i]; j++){

```

```

41         if(j!=4&&!(j==2&&v[i+1]==6))
42             res+=dp[i][j];
43     }
44     if(v[i]==4)
45         break;
46     if(v[i]==2&&v[i+1]==6)
47         break;
48     }
49     return res;
50 }

```

## 51 2、Beautiful numbers(<https://codeforces.com/problemset/problem/55/D>)

52 题意:

53 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。

54

55 思路:

56 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。

57 2、存在定理 $a\%(x*n)\%x=a\%x$

58 3、[1,9]所有的组合的最小公倍数，最大值为2520，共有48个， $2520\%lcm[\text{任意组合}]=0$ 。

59 综上所述，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

60 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

61 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。

62

63 核心代码:

```

64 ll dfs(int pos, int tot, int lcm, bool limit) {
65     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
66         return (tot%lcm == 0);
67     if (!limit&&dp[pos][ha[lcm]][tot] != -1)
68         return dp[pos][ha[lcm]][tot];
69     ll res = 0;
70     int top = limit ? di[pos] : 9; //判断当前有限制
71     for (int i = 0; i <= top; i++) {
72         res += dfs(pos - 1, (tot * 10 + i) % mod, i ? i * lcm / gcd(i, lcm) : lcm, i == di[
73             pos] && limit);
74     } //进一步搜索
75     if (!limit) //如果没有限制,那么这个情况的值就确定下来了
76         dp[pos][ha[lcm]][tot] = res;
77     return res;
78 }

```

79

## 80 3、Beautiful numbers(<https://ac.nowcoder.com/acm/contest/163/J>)

81 题意:

82 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 $1 \leq n \leq 1e12$

83 思路:

84 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。

85 如果存在 $\sigma(di) \equiv mod$ ，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

86 余k的数量。

87

88 核心代码:

```

89 ll dfs(int pos, int tot, int remain, bool limit) {
90     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
91         return (tot == mod && !remain);
92     if (!limit&&dp[pos][tot][remain] != -1)
93         return dp[pos][tot][remain];

```

```

94     ll res = 0;
95     int top = limit ? di[pos] : 9;
96     for (int i = 0; i <= top; i++) {
97         if (i + tot > mod)
98             break;
99         res += dfs(pos - 1, i + tot, (10 * remain + i) % mod, limit && i == di[pos]);
100    }
101    if (!limit)
102        dp[pos][tot][remain] = res;
103    return res;
104 }

```

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,但可以通过记忆化搜索进行剪枝,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( $1 \leq n \leq 1000000000$ ),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:

```

118 ll dfs(int pos, int tot, int state, bool limit) {
119     if (pos == -1)
120         return (state == 2 && !tot);
121     if (!limit && dp[pos][tot][state] != -1)
122         return dp[pos][tot][state];
123     int top = limit ? di[pos] : 9;
124     ll res = 0;
125     for (int i = 0; i <= top; i++) {
126         int cstate = state;
127         if (state == 1 && i == 3)
128             cstate = 2;
129         else if (state == 1 && i != 3 && i != 1)
130             cstate = 0;
131         else if (state == 0 && i == 1)
132             cstate = 1;
133         res += dfs(pos - 1, (tot * 10 + i) % 13, cstate, limit && di[pos] == i);
134     }
135     if (!limit)
136         dp[pos][tot][state] = res;
137     return res;
138 }

```

反思:

最开始,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

146 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.

147

## 148 5、F(x) (hdu4734)

149 题意:

150 存在一个函数 $F(x)=A_n*2^{(n-1)}+A_{n-1}*2^{(n-2)}+\dots+A_1*1$ , 其中 $x=(A_nA_{n-1}A_{n-2}\dots A_1)$ , 现在求 $x=[0, B]$ 中, 求 $F(x)\leq F(A)$ 的个数

151

152 思路:

153 显然是一道数位dp题, 这题的 $t(1\leq t\leq 10000)$ , 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

154 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

155 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

156 状态不正确.

157

158 核心代码:

```
159 ll dfs(int pos, int tot, bool limit) {
160     if (pos == -1)
161         return tot>=0;
162     if (!limit&&dp[pos][tot] != -1)
163         return dp[pos][tot];
164     int res = 0;
165     int top = limit ? di[pos] : 9;
166     for (int i = 0; i <= top; i++) {
167         int ntot = tot - i * (1 << pos);
168         if (ntot < 0)
169             break;
170         res += dfs(pos - 1, ntot, limit&&di[pos] == i);
171     }
172     if (!limit)
173         dp[pos][tot] = res;
174     return res;
175 }
```

176

177 反思:

178 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.

179

## 180 6、Balanced Number(hdu 3709)

181 题意:

182 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

183

184 思路:

185 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

186 tot==0, 说明存在平衡.

187

188 核心代码:

```
189 ll dfs(int pos, int tot, int pivot, bool limit) {
190     if (pos == -1)
191         return tot==0;
192     if (!limit&&dp[pos][pivot][tot] != -1)
193         return dp[pos][pivot][tot];
194     ll res = 0;
195     int top = limit ? di[pos] : 9;
196     for (int i = 0; i <= top; i++) {
197         int tmp = tot + i * (pos - pivot);
198         if (tmp < 0)
```

```

199         break;
200         res += dfs(pos - 1, tmp, pivot, limit && di[pos] == i);
201     }
202     if (!limit)
203         dp[pos][pivot][tot] = res;
204     return res;
205 }

```

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

平衡数。

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 $3 \times 10$ ,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:

```

220 ll dfs(int pos, int state, bool limit){
221     if(pos == -1)
222         return judge(state);
223     if(!limit && dp[pos][state] != -1)
224         return dp[pos][state];
225     ll res = 0;
226     int top = limit ? di[pos] : 9;
227     for(int i = 0; i <= top; i++){
228         res += dfs(pos - 1, (state == 0 && i == 0) ? 0 : change(state, i), limit && di[pos] == i);
229     }
230     if(!limit)
231         dp[pos][state] = res;
232     return res;
233 }

```

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FFFFFFF,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。



250

251 8、吉哥系列故事——恨7不成妻(hdu4507)

252 题意：

253 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

254 的整数倍。3、这个整数不是7的整数倍。

255

256 思路：

257 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

258 关键发现 $(x_1+x_2+\dots+x_n)^2=x_1^2+2*x_1*(x_2+\dots+x_n)+(x_2+\dots+x_n)^2=x_1^2+2*x_1*(x_2+\dots+x_n)+x_2^2+2*x_2*(x_3+\dots+x_n)+$ 259  $(x_3+\dots+x_n)^2=\dots$  无限递归下去。

260

261 核心代码：

```

262 node dfs(int pos, int tot, int tot2, bool limit) {
263     node res; res.cnt = res.sqrsum = res.sum = 0;
264     if (pos == -1) {
265         res.cnt = (tot != 0 && tot2 != 0);
266         return res;
267     }
268     if (!limit && dp[pos][tot][tot2].cnt != -1)
269         return dp[pos][tot][tot2];
270     int top = limit ? di[pos] : 9;
271     for (int i = 0; i <= top; i++) {
272         if (i == 7)
273             continue;
274         node tmp = dfs(pos - 1, (tot + i) % 7, (tot2 * 10 + i) % 7, limit && di[pos] == i);
275         ll A = p[pos] * i % mod;
276         res.cnt = (res.cnt + tmp.cnt) % mod;
277         res.sum = (res.sum + tmp.sum + A * tmp.cnt % mod) % mod;
278         res.sqrsum = (res.sqrsum + A * A % mod * tmp.cnt % mod + tmp.sqrsum + 2 * A % mod * tmp.sum % mod) % mod;
279     }
280     if (!limit)
281         dp[pos][tot][tot2] = res;
282     return res;
283 }

```

284

285 反思：

286 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

287 保存更多状态。

288

289 9、pair(<https://ac.nowcoder.com/acm/contest/887/H>)

290 题意：

291 给你A,B,C,现在求有多少个(x,y)满足 $x \& y > c$  ||  $x \wedge y < c$ , 其中( $1 \leq x \leq A, 1 \leq y \leq B, 1 \leq A, B, C \leq 1e9$ )

292

293 思路：

294 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 $A*B - \text{cnt}(x \& y \leq c \& x \wedge y \geq c) + \max(0, A - C + 1) + \max(0, B - C + 1)$ ，因为x,y

295 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，

296 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。

297

298 代码：

```

299 ll dfs(int len, int AND, int XOR, int limit1, int limit2) {
300     if (len < 0)
301         return 1;

```



```

302     if(dp[len][AND][XOR][limit1][limit2]!=-1) return dp[len][AND][XOR][limit1][limit2];
303     int top=limit1?(A>>len)&1:1;
304     int top2=limit2?(B>>len)&1:1;
305     int c=(C>>len)&1;
306     ll cnt=0;
307     for(int i=0;i<=top;i++){
308         for(int j=0;j<=top2;j++){
309             if(((!AND||((i&j)<=c)&&(!XOR||((i^j)>=c)))){
310                 cnt+=dfs(len-1,AND&&(i&j)==c,XOR&&(i^j)==c,limit1&&i==top,limit2&&j==
top2);
311             }
312         }
313     }
314     return dp[len][AND][XOR][limit1][limit2] = cnt;
315 }
316
317 int main(){
318     int t;
319     cin>>t;
320     while(t--){
321         cin>>A>>B>>C;
322         memset(dp,-1,sizeof(dp));
323         cout<<(A*B-dfs(30,1,1,1,1)+max(0ll,A-C+1)+max(0ll,B-C+1))<<endl;
324     }
325     return 0;
326 }

```

## 7.4 概率 dp

- 1 规律总结:
- 2 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 $E(aA+bB\dots)=aE(A)+bE(B)+\dots+1$
- 3 2、期望从后往前找, 一般 $dp[n]=0$ ,  $dp[0]$ 是答案
- 4 3、解决过程, 找出各种情况乘上这种情况发生的概率, 求和
- 5
- 6 1、Favorite Dice (spoj)
- 7 题意:
- 8 甩一个n面的骰子, 问每一面都被甩到的次数期望是多少?
- 9
- 10 思路:
- 11 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.
- 12 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.
- 13
- 14 核心代码:
- 15  $dp[n] = 0;$
- 16  $\text{for} (\text{int } i = n - 1; i \geq 0; i--)$
- 17  $\quad dp[i] = dp[i + 1] + 1.0*n / (n - i);$
- 18  $\text{printf}("%.21f\n", dp[0]);$
- 19
- 20 2、LOOPS (hdu3853)
- 21 题意:
- 22 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.
- 23
- 24 思路:
- 25 设 $dp[r][c]=0$ , 那么有满足状态转移方程  $dp[i][j] = p1*dp[i][j]+p2*dp[i][j+1]+p3*dp[i+1][j]+2$

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

27

28 核心代码:

```
29 for (int i = r; i >= 1; i--) {
30     for (int j = c; j >= 1; j--) {
31         if (i == r && j == c)
32             continue;
33         if (p[i][j].p1 == 1)
34             continue;
35         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);
36     }
37 }
```

38

39 反思:

40  $dp[i][j]$ 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。  
 41 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  
 42 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。

43

### 44 3、King Arthur's Birthday Celebration(poj3682)

45 题意:

46 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

47 结束宴会的期望天数和期望花钱量。

48

49 思路:

50 设 $dp[i]$ 表示投中 $i$ 次后, 还需要天数的期望。显然满足  $dp[i]=(1-p)*dp[i]+p*dp[i+1]+1 \Rightarrow dp[i]=dp[i+1]+1/p$ 。  
 51  $dp2[i]$ 表示投中 $i$ 次后, 还需要花钱的期望, 显然满足  $dp2[i]=(1-p)*(dp2[i]+2*(dp[i]+1)-1)+p*(dp2[i+1]+2*(dp[i+1]+1)-1)$ 。  
 52  $\Rightarrow$  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。

53 也可以正推, 显然满足  $dp[i]=(1-p)*dp[i]+p*dp[i-1]+1 \Rightarrow dp[i]=dp[i-1]+1/p$ ,  
 54  $dp2[i]=(1-p)*(dp2[i]+2*dp[i]-1)+p*(dp2[i+1]+2*dp[i+1]-1)+2$ 。然后 $dp[0]=dp2[0]=0$ , for循环一下就可以得到答案。

55

56 反思:

57 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。

58

### 59 4、烟花 (<https://ac.nowcoder.com/acm/contest/180/B>)

60 题意:

61 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 $p_i$ 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。

62

63 思路:

64 第一个就是简单期望相加, 即 $E(x)=\sum p_i$

65 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 $dp[0][0]=1, dp[0][i]=(1-p[1])(1-p[2])\dots(1-p[i])$ 。那么就有 $dp[i][j]=dp[i-1][j-1]*p[j]+dp[i][j-1]*(1-p[j])$ , 那么  
 66  $dp[k][n]$ 即为所求答案。

67 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 $j \leq \min(i, k)$ 即可。

72

### 73 5、流星雨(<https://ac.nowcoder.com/acm/contest/368/C>)

74 题意:

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

76 那么第二天有流星雨的可能性是 $p_2+P$ , 否则是 $p_2$ 。相应的, 如果第 $i-1$  ( $i \geq 2$ )天有流星雨, 第 $i$ 天有流星雨的可能性是

77  $p_i+P$ , 否则是 $p_i$ 。求 $n$ 天后, 流星雨颗数的期望。

78

79 思路:

80 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:  $dp[i]=dp[i-1]*(p_i+p)+(1-dp[i-1])*p$ 。然后只需乘上每天的流星雨数量即可得到答案。

81

82

83 反思:

84 这道题我只想着如何直接推导期望公式, 导致推出来的公式非常耗时间, 公式如下:

85  $dp[i]=dp[i-1]+(p^{i-1}*p[1]+\dots+p^0*p[i])*w[i]$ 。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

86 即 $E(x)=p[x]*w[x]$ , 我们分段求出期望最后累加就是结果。

87

88 6、One Person Game(zoj3329)

89 题意:

90 有三个骰子, 分别有 $k_1, k_2, k_3$ 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。

91 当分数大于 $n$ 时结束。求游戏的期望步数。初始分数为0

92

93 思路:

94 设 $dp[i]$ 表示达到 $i$ 分时到达目标状态的期望,  $p_k$ 为投掷 $k$ 分的概率,  $p_0$ 为回到0的概率则 $dp[i]=\sum(p_k*dp[i+k])+dp[0]*p_0+1$ ;

95 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:

96  $dp[i]=\sum(p_k*A[i+k]*dp[0]+p_k*B[i+k])+dp[0]*p_0+1=(\sum(p_k*A[i+k])+p_0)dp[0]+\sum(p_k*B[i+k])+1$ ;明显

97  $A[i]=\sum(p_k*A[i+k])+p_0$

98  $B[i]=\sum(p_k*B[i+k])+1$ , 先递推求得 $A[0]$ 和 $B[0]$ 。那么 $dp[0]=B[0]/(1-A[0])$ ;

99

99 7、Dice(hdu 4652)

100 题意:

101 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

102 数字的期望。

103

104 思路:

105 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

106  $dp[0]-dp[1]=1, dp[1]-dp[2]=m, \dots, dp[n-1]-dp[n]=m^{n-1}$ , 可以得到 $dp[0]=(m^n-1)/(m-1)$

107 对于2情况, 要注意是连续 $n$ 个数不相同, 因此 $dp[n]=0, dp[i]=1+(m-i)*dp[i]/m+(dp[1]+dp[2]+\dots dp[i])/m$ , 因为如果加入相同的数, 那么

108 会随着数字的不同回到原始点不同, 因此有了 $(dp[1]+dp[2]+\dots dp[i])/m$ , 方法如上递推即可。

109

110 8、Maze(hdu 4035)

111 题意:

112 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 $k_i$ )

113 2. 找到出口, 走出迷宫 (概率为 $e_i$ ) 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。

114

115 思路:

116 设 $dp[i]=A_i*dp[1]+B_i*dp[fa[i]]+C_i, re[i]=1-k_i-e_i, other=re[i]/m$

117 关于叶子节点:  $dp[i]=k_i*dp[1]+E_i*0+other*(dp[fa[i]]+1)=k_i*dp[1]+other*dp[fa[i]]+other$ , 因此可以得到叶子节点:

118  $A_i=k_i, B_i=other, C_i=other$

119 关于非叶子节点:  $dp[i]=k_i*dp[1]+E_i*0+other*(dp[fa[i]]+1+\sigma(dp[child[i]]+1))$ , 设 $j=nx[i]$ , 则 $\sigma(dp[child[i]]+1)$

120  $=\sigma(A_j)*dp[1]+\sigma(B_j)*dp[fa[j]]+\sigma(C_j)$ , 带入原式可以得到 $A_i=(k_i+other*\sigma(A_j))/(1-other*\sigma(B_j))$ ,

121  $B_i=other/(1-other*\sigma(B_j)), C_i=(re[i]+other)/(1-other*\sigma(B_j))$

122 因此一个dfs即可。

```

123
124 代码:
125 bool dfs(int from,int pre){
126     int m = g[from].size();
127     double other = re[from]/m;
128     double a = k[from],b = 1,c = re[from];
129     for(int i=0;i<m;i++){
130         int to = g[from][i];
131         if(to==pre) continue;
132         if(!dfs(to,from)) return false;
133         a+=other*A[to]; b-=other*B[to]; c+=other*C[to];
134     }
135     if(b<eps) return false;
136     A[from]=a/b; B[from]=other/b; C[from]=c/b;
137     return true;
138 }
139
140 9.Activation
141 题意:
142     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
143 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
144 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。
145
146 思路:
147     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。
148     当j=1时, dp[i][j] = p1*dp[i][j]+p2*dp[i][i]+p4;
149     当1<j<=k时, dp[i][j]= p1*dp[i][j]+p2*dp[i][j-1]+p3*dp[i-1][j-1]+p4;
150     当k<j<=i时, dp[i][j] = p1*dp[i][j]+p2*dp[i][j-1]+p3*dp[i-1][j-1];
151     化简得:
152     当j=1时, dp[i][j] = p21*dp[i][i]+p41;
153     当1<j<=k时, dp[i][j]= p21*dp[i][j-1]+p31*dp[i-1][j-1]+p41;
154     当k<j<=i时, dp[i][j] = p21*dp[i][j-1]+p31*dp[i-1][j-1];
155     其中: p21=p2/(1-p1)、p31=p3/(1-p1)、p41=p4/(1-p1);
156     循环i: 1-n;
157     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:
158     当j=1时, dp[i][j] = p21*dp[i][i]+c[1];
159     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;
160     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];
161     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]:
162     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]
163     得出dp[i][i],那么dp[i][1]也可以得出,之后就递推就行了。
164
165 ///////////////////////////////////////////////////
166
167 3、Kids and Prizes(sgu495)
168 题意:
169     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,
170     然后把盒子放回原处。为最后m个人取走礼物的期望。
171
172 思路:
173 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)
174 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:

```

175  $dp[i] = dp[i-1] + (n-dp[i-1])/n*1+dp[i-1]/n*0 \Rightarrow dp[i] = dp[i-1] + (n-dp[i-1])/n$

176

177 反思:

178 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

179 可以尝试使用日常经验的手段去推。

180

181

182

183 5、Where is the canteen(hdu2262)

184 题意:

185 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

186

187 思路:

188 显然此题满足概率dp, 状态转移方程满足  $dp[i][j]=p1*dp[i+1][j]+p2*dp[i-1][j]+p3*dp[i][j-1]+p4*$

189  $dp[i][j+1]+1$ , 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

190 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

191 四个防线的  $dp[i][j]*p$ +移动步数1即可。

192

193 反思: 代码不够优美, 要多看看优美的代码

## 7.5 斜率 dp

1 1、Print Article(hdu 3507)

2 题意:

3 给出N个单词, 每个单词有个非负权值  $C_i$ , 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,

4 即  $(\sum C_i)^2 + M$ 。现在想求出一种最优方案, 使得总费用之和最小。

5

6 思路:

7 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。

8

9 2、Lawrence(hdu2829)

10 题意:

11 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, 现在在m次攻击机会, 一次攻

12 击可以炸毁一段铁路; m次攻击后, 剩余的总价值为:  $\sum(v[i]*v[j])$ , i和j为所有任意两个互相可达的仓库。现要求选定m段铁路进行攻击炸毁,

13 然后使得总价值最小。

14

15 思路:

16 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,

17 那么, 就有:  $dp[i][j]=\min(dp[k][j-1]+w[k+1][i]), j \leq k < i$ ; 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

18 价值, 枚举k找到最小的。

19 那么如何计算  $w[k+1][i]$  呢?  $w[1][i]=w[1][k]+w[k+1][i]+(v[1]+v[2]+\dots+v[k]) \times (v[k+1]+v[k+2]+\dots+v[i])=w[1][k]+w[k+1][i]+$

20  $\sum[k] \times (\sum[i]-\sum[k])$ , 即  $w[k+1][i]=w[1][i]-w[1][k]-\sum[k] \times (\sum[i]-\sum[k])$

21 我们把  $w[k+1][i]$  的计算式带入状态转移方程得到:  $dp[i][j]=\min\{dp[k][j-1]+w[1][i]-w[1][k]-\sum[k] \times (\sum[i]-\sum[k])\}$

22 那么, 对于这个DP, j一个循环、i一个循环、k一个循环, 就是  $O(n^3)$  的时间复杂度; 需要斜率优化, 优化到  $O(n^2)$  即可。

23

24 3、Cross the Wall(Uvalive 5097)

25 题意:

26 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

27 最小代价

28

29 思路：

30 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

31 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。

32

33 代码：

34 `sort(p+1,p+1+n,cmp);`

35 `int tot=1;`

36 `for(int i=2;i<=n;i++){`

37 `if(p[i].w<=p[tot].w) continue;`

38 `p[++tot]=p[i];`

39 `}`

40 `n=tot;`

41 `for(int i=1;i<=n;i++) dp[1][i]=p[1].h*p[i].w;`

42 `ll ans=dp[1][n];`

43 `m=min(m,n);`

44 `for(int k=2;k<=m;k++){`

45 `int tail=0,head=0;`

46 `q[tail++]=0; dp[k][k]=0;`

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

48 `while (tail>1+head&&isOK(k,i,q[head+1],q[head])) head++;`

49 `dp[k][i]=getsum(k,i,q[head]);`

50 `while (tail>1+head&&isOK2(k,i,q[tail-1],q[tail-2])) tail--;`

51 `q[tail++]=i;`

52 `}`

53 `ans=min(ans,dp[k][n]);`

54 `}`

55 `printf("%lld\n",ans);`

56

57 4、Picnic Cows(hdu3045)

58 题意：

59 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

60 价值牛的差的和，问所有组贡献的价值最小是多少。

61

62 思路：

63 这题dp方程容易想出 $dp[i]=dp[k]+sum[i]-sum[k]-(i-k)*a[k+1]$ ; $(k \geq m \ \&\& \ i-(k+1)+1 \geq m)$ ，但是在  
处理的时候有一定的技巧

64 `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;`

65

66 5、Tree Construction(hdu3516)

67 题意：

68 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

69

70 思路：

71  $f[i][j]$ 表示把 $[i, j]$ 中的点合成一棵树的最小代价， $f[i][j]=\min\{f[i][k]+f[k+1][j]+abs(x[k+1]-x[i])+abs(y[k]-y[j])\}$

72 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 $s[i][i]=i$

73

74 6、Post Office(1160)

75 题意：

76 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，问如何选择，才能使每个村庄  
到其最近邮局的距离

77 和最小？最后打印这个最小值。

```

78
79 思路:
80     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
81 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
82 for(n->i),并且需要注意越界的情况in[i][v+1]=v-1;
83
84 代码:
85 int main() {
86     while(scanf("%d%d",&v,&p)==2){
87         for(int i=1;i<=v;i++){
88             scanf("%lld",&a[i]);
89             s[i]=s[i-1]+a[i];w[i][i]=0;in[i][i]=i;
90         }
91         for(int len=2;len<=v;len++){
92             for(int i=1;i+len-1<=v;i++){
93                 int j=i+len-1;
94                 w[i][j]=INF; int index;
95                 for(int k=in[i][j-1];k<=in[i+1][j];k++){
96                     ll x = a[k]*(2*k-i-j+1)+s[j]+s[i-1]-2*s[k];
97                     if(w[i][j]>x){
98                         w[i][j]=x; index=k;
99                     }
100                 }
101                 in[i][j]=index;
102             }
103         }
104         for(int i=1;i<=v;i++) dp[1][i]=w[1][i],in[1][i]=1;
105         for(int i=2;i<=p;i++){
106             in[i][v+1]=v-1; dp[i][i]=0;
107             for(int j=v;j>=i+1;j--){
108                 dp[i][j]=INF;
109                 for(int k=in[i-1][j];k<=in[i][j+1];k++){
110                     ll x = dp[i-1][k]+w[k+1][j];
111                     if(dp[i][j]>=x) dp[i][j]=x,in[i][j]=k;
112                 }
113             }
114         }
115         printf("%lld\n",dp[p][v]);
116     }
117 }

```

## 119 7、Batch Scheduling(poj1180)

120 题意:

121 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,  
 122 这些任务被分批加工,第i个任务单独完成所需的时间是 $T_i$ .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  
 123 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 $F_i$ 。请确定一个  
 分组方案,使得  
 124 总费用最小。(1<=N<=10000)

125 思路:

127 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  $f[i][j]=\text{Min}\{f[i-1][k]+(S*i+T[j])*(c[j]-c[k])\}$ ;  
 128 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。  
 129  $\text{sumT}[i]$ 表示从i到n的任务所需要的时间总和,  $\text{sumF}[i]$ 表示从i到n的费用系数总和,  $\text{dp}[i]$ 表示对于从i到n的  
 任务安排的最优解。

130 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)  
 131  $dp[i] = \min\{dp[j] + (S + sumT[i] - sumT[j]) * sumF[i]\} \{i < j \leq n+1\}$  边界条件  $dp[n+1] = 0$   
 132 从后往前推效率可以降低一维的原因：  
 133 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。  
 134 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。  
 135  
 136 8、Best Cow Fences(poj2018)  
 137 题意：  
 138 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。  
 139  
 140 思路：  
 141 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， $b[i] = a[i] - mid$   
 142 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  
 143 区间和=前缀和-前面的最小前缀和(要保证区间长度大于F)，判断和是否大于等于0  
 144 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  
 145  
 146 代码：  
 147 

```
bool isOK(double x){
  148     for(int i=1;i<=n;i++) s[i]=s[i-1]+cow[i]-x;
  149     double minn = inf;
  150     for(int i=f;i<=n;i++){
  151         minn=min(minn,s[i-f]);
  152         if(s[i]-minn>=0) return true;
  153     }
  154     return false;
  155 }
```

## 7.6 树形 dp

1 1、Computer(hdu 2196)  
 2 题意：  
 3 给出一棵树，求离每个节点最远的点的距离。  
 4  
 5 思路：  
 6 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  
 7 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，  
 8 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  
 9 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，  
 10 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  
 11 距离。  
 12 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，  
 13 设 $f[i][2]$ 为i节点的父亲部的最远  
 14 点距离。那么我们列出dp方程：  
 15 当x不在fa的最远点路径上： $f[x][2] = \max(f[fa][0], f[fa][2]) + \text{dist}(x, fa)$   
 16 当x在fa的最远点路径上： $f[x][2] = \max(f[fa][1], f[fa][2]) + \text{dist}(x, fa)$   
 17  
 18 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 $\max(\text{dis}[i], \text{dis2}[i])$ 即可  
 19  
 20 代码：



```

21 int n,dp[N][3],p[N][2];
22 vector<pii>e[N];
23 void dfs(int from,int pre){
24     dp[from][0]=dp[from][1]=0;
25     for(int i=0;i<e[from].size();i++){
26         int to = e[from][i].first,w=e[from][i].second;
27         if(pre==to) continue;
28         dfs(to,from);
29         if(dp[from][0]<dp[to][0]+w){
30             dp[from][1]=dp[from][0];
31             dp[from][0]=dp[to][0]+w;
32             p[from][0]=to;
33         }else if(dp[from][1]<dp[to][0]+w){
34             dp[from][1]=dp[to][0]+w;
35             p[from][1]=to;
36         }
37     }
38 }
39
40 void dfs2(int from,int pre){
41     for(int i=0;i<e[from].size();i++){
42         int to = e[from][i].first,w=e[from][i].second;
43         if(pre==to) continue;
44         if(to!=p[from][0]) dp[to][2]=w+max(dp[from][2],dp[from][0]);
45         else dp[to][2]=w+max(dp[from][2],dp[from][1]);
46         dfs2(to,from);
47     }
48 }

```

## 50 2、Rebuilding Roads(poj1947)

51 题意:

52 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？

53 思路:

54 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

55 代码:

```

56 void dfs(int from,int pre){
57     memset(dp[from],0x3f,sizeof(dp[from])); dp[from][1]=0; node[from]=1;
58     for(int i=0;i<e[from].size();i++){
59         int to=e[from][i];
60         if(pre==to) continue;
61         dfs(to,from);
62         node[from]+=node[to];dp[from][1]++;
63     }
64     for(int i=0;i<e[from].size();i++){
65         int to=e[from][i];
66         if(pre==to) continue;
67         for(int j=m;j>=1;j--){
68             for(int k=1;k<=node[to]&& j-k>=1;k++){
69                 dp[from][j]=min(dp[from][j],dp[from][j-k]+dp[to][k]-1);
70             }
71         }
72     }
73 }

```

## 75 3、Starship Troopers(hdu1011)

76 题意:

77 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，

78 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 $dp[son][0], dp[son][1], \dots$

$dp[son][k]$ , 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;
        }
    }
}
```

```

129         for(int k=1;k<=j;k++)
130             dp[from][j]=min(dp[from][j],dp[from][j-k]+dp[to][k]+k*w);
131     }
132 }
133 }

```

#### 134 5、The Ghost Blows Light(hdu4276)

136 题意：

137 一个有N个节点的树形的地图，知道了每条边经过所需要的时间，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。  
138 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。

139

140 思路：

141 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  
142 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  
143 还要再走回来，状态转移方程：

```

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

```

145

#### 146 6、Fire(poj2152)

147 题意：

148 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

149 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

150 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。

151

152 思路：

153  $dp[i][j]$ 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

154 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)

155  $dis[i]$ 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。

156 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

157 站到他最远距离)，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

158 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

159 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。

160

161 代码：

```

162 int n,w[N],d[N],head[N],tot,best[N],dis[N];
163 int dp[N][N];
164 struct node{
165     int to,nx,w;
166 }edge[N<<1];
167
168 void add_edge(int from,int to,int val){
169     edge[tot].to=to; edge[tot].nx=head[from]; edge[tot].w=val;
170     head[from]=tot++;
171 }
172
173 void DFS(int from){
174     for(int i=head[from];i;i=edge[i].nx){
175         int to=edge[i].to;
176         if(dis[to]!=-1) continue;

```

```

177         dis[to]=dis[from]+edge[i].w; DFS(to);
178     }
179 }
180
181 void dfs(int from,int pre){
182     for(int i=head[from];i;i=edge[i].nx){
183         int to=edge[i].to;
184         if(to==pre) continue;
185         dfs(to,from);
186     }
187     memset(dis,-1, sizeof(int)*(n+1));
188     dis[from]=0;DFS(from);
189     best[from]=inf;
190     for(int i=1;i<=n;i++) dp[from][i]=inf;
191     for(int i=1;i<=n;i++){
192         if(dis[i]<=d[from]){
193             dp[from][i]=w[i];
194             for(int j=head[from];j;j=edge[j].nx){
195                 int to=edge[j].to;
196                 if(to==pre) continue;
197                 dp[from][i]+=min(best[to],dp[to][i]-w[i]);
198             }
199             best[from]=min(best[from],dp[from][i]);
200         }
201     }
202 }
203
204 int main() {
205     int t,x,y,z;
206     scanf("%d",&t);
207     while(t--){
208         scanf("%d",&n);
209         for(int i=1;i<=n;i++) scanf("%d",&w[i]);
210         for(int i=1;i<=n;i++) scanf("%d",&d[i]);
211         memset(head,0, sizeof(int)*(n+1)); tot=1;
212         for(int i=1;i<n;i++){
213             scanf("%d%d%d",&x,&y,&z);
214             add_edge(x,y,z); add_edge(y,x,z);
215         }
216         dfs(1,0);
217         printf("%d\n",best[1]);
218     }
219     return 0;
220 }

```

## 222 7、GeoDefense(hdu4044)

223 题意:

224 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，

225 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 $k_i$ 种塔供你选择，价钱和攻击力分别为

226  $price_i$ ,  $power_i$ , 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

227 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，

228 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

229 基地中抵抗力最低的一个的值尽量大，最大是多少？

230

```

231 思路:
232 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,
233 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
234
235 代码:
236 int n,m;
237 vector<int>e[N];
238 int dp[N][M],k[N];
239 struct node{
240     int price,val;
241 }p[N][505];
242
243 void dfs(int from,int pre){
244     dp[from][0]=inf;
245     for(int i=0;i<e[from].size();i++){
246         int to=e[from][i];
247         if(to==pre) continue;
248         dfs(to,from);
249         for(int j=m;j>=0;j--){
250             int x=0;
251             for(int k=0;k<=j;k++) x=max(x,min(dp[to][k],dp[from][j-k]));
252             dp[from][j]=x;
253         }
254     }
255     if(dp[from][0]==inf) dp[from][0]=0;
256     for(int i=m;i>=0;i--){
257         int x=dp[from][i];
258         for(int j=1;j<=k[from];j++){
259             if(p[from][j].price<=i){
260                 x=max(x,dp[from][i-p[from][j].price]+p[from][j].val);
261             }
262         }
263         dp[from][i]=x;
264     }
265 }
266
267 int main() {
268     int t,u,v;
269     scanf("%d",&t);
270     while(t--){
271         scanf("%d",&n);
272         for(int i=1;i<=n;i++) e[i].clear();
273         for(int i=1;i<=n;i++){
274             scanf("%d%d",&u,&v);
275             e[u].push_back(v); e[v].push_back(u);
276         }
277         scanf("%d",&m);
278         for(int i=1;i<=n;i++){
279             scanf("%d",&k[i]);
280             for(int j=1;j<=k[i];j++){
281                 scanf("%d%d",&p[i][j].price,&p[i][j].val);
282             }
283         }
284         for(int i=1;i<=n;i++) for(int j=1;j<=m;j++) dp[i][j]=0;
285         dfs(1,0);
286         printf("%d\n",dp[1][m]);
287     }
288     return 0;

```

289 }

## 7.7 KMP、E-KMP、Manacher

1 KMP内容:

2

3 1、KMP最小循环节、循环周期:

4

5 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]。

6

7 (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。

8

9 (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]。

10

11 注意: 对于补全最小循环节,  $n \% (n - \text{Next}[n]) == 0 \ \&\& \ n / (n - \text{Next}[n]) > 1$  表示无需再补循环节,  $n - \text{Next}[n] - n \% (n - \text{Next}[n])$ 表示

12 补全最小循环节的最少个数。

13

14 1) 求最小循环节周期>1的情况

```
15 for(int i=1;i<=n;i++){
16     if(i%(i-nx[i])==0&&i/(i-nx[i])>1)
17         printf("%d %d\n",i,i/(i-nx[i]));
18 }
```

19

20 2) 求最小循环节周期

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

22

23 3) 补全最小循环节, 周期要K>=2

```
24 if (n % (n - Next[n]) == 0 && n / (n - Next[n]) > 1){
25     printf("0\n");
26     continue;
27 }
```

28

```
29 int len = n - Next[n]; // 最小循环节
```

```
30 printf("%d\n", len - n % len);
```

31

32 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。

```
33 int KMP() {
34     int i, j;
35     getNext();
36     i = j = 0;
37     while (j < n&&i<n) {
38         while (i != -1 && x[i] != y[j])
39             i = nx[i];
40         i++;j++;
41     }
42     return 2*n-i;
43 }
```

44

45 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

```
47 for(int i=1;i<=n;i++){
48     dp[i]=(dp[nx[i]]+1)%mod;
49     res=(res+dp[i])%mod;
50 }
```

51

52 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。

53 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

54 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。

```
55 for(int i=0; i<m; i++){
56     for(int j=0; j<n; j++){
57         int t = i;
58         while(t>0&&y[j]!=x[t])
59             t=nx[t];
60         if(y[j]==x[t]||t==-1)
61             t++;
62         A.arr[i][t]++;
63     }
64 }
```

66 5、给定一个字符矩阵( $1 \leq r \leq 10000, 1 \leq c \leq 75$ )，求最小的子矩阵可以将矩阵全部覆盖，运行多余。

67 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，

```
68 Input
69 2 8
70 ABCDEFAB
71 ABCDEABC
72 2 8
73 ABCDEFAB
74 AAAABAAA
75 Output
76 16
77 12
```

78 显然上述方法存在问题，因此得寻找其他方法。注意到 $1 \leq c \leq 75$ ，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，

79 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

80 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。

```
81 void solve_col(){
82     int i, j;
83     j=nx[0]=-1;
84     i=0;
85     while(i<r){
86         while(-1!=j&&strcmp(cow[i], cow[j])!=0)
87             j=nx[j];
88         nx[++i]=++j;
89     }
90     ans_row=r-nx[r];
91 }
```

93 E-KMP内容:

94 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： $extend[i]$

95 表示 $T$ 与 $S[i, n-1]$ 的最长公共前缀，要求出所有 $extend[i]$  ( $0 \leq i < n$ )。

96 (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。)

97

98 1、求一个字符串所有前缀在字符串中出现次数和

99 对该字符串进行e-kmp，得到 $nx[n]$ 数组，显然 $sum(nx[0] \dots nx[n-1])$ 就是答案。



## 7.8 后缀数组、后缀树、后缀自动机

```

1  后缀数组:
2      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
3  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,
4  rk表示第i个的排名是啥.
5      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)
6  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))
7  对于任意1<=j<=k<=n
8      如何求LCP?
9      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
10 设h[i]=height[rk[i]], 同样的, height[i]=h[sa[i]]; => h[i]>=h[i-1]-1;
11
12 应用:
13 1)求两个字符串最大的公共子串
14      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height, 只需要扫一遍height即可得到最优结
      果. height[i]为LCP(i, i-1),
15 1<=i<=n, LCP(i, j)为suff(sa[i])与suff(sa[j])的最长公共前缀.
16 for(int i=1; i<=n; i++){
17     if(sa[i]<p&&sa[i-1]>p) res = max(res, height[i]);
18     if(sa[i]>p&&sa[i-1]<p) res = max(res, height[i]);
19 }
20
21 2) 求一个串中所有子串的种类个数
22      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.
23 而height[i]=LCP(i-1, i)的最长公共前缀.
24 ll res=0;
25 for(int i=2; i<=n; i++){
26     int top = height[i];
27     res+=(n-sa[i-1]+1-top);
28 }
29 res+=(n-sa[n]+1);
30
31 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
32      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]
33 更新区间长度最大值即可.
34
35 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
36      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.
37 bool isOK(int n, int k){
38     vector<pii>v;
39     for(int i=2; i<=n; i++){
40         if(height[i]+1>=k){
41             int j;
42             for(j=i; j<=n; j++){
43                 if(height[j]+1<k) break;
44                 int l1=sa[j-1], r1=sa[j-1]+k-1;
45                 int l2=sa[j], r2=sa[j]+k-1;
46                 if(check(l1, r1, l2, r2)) return true;
47             }
48             for(int k=0; k<v.size(); k++){
49                 if(check(l2, r2, v[k].first, v[k].second)) return true;

```



```

50         v.push_back(make_pair(l1,r1));
51     }
52     i=j; v.clear();
53 }
54 }
55 return false;
56 }
57

```

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(摘自罗穗骥的国家集训队论文): 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 $r[0], r[L], r[L*2], r[L*3], \dots$ 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 $O(n/1+n/2+n/3+\dots+n/n)=O(n \log n)$ 。

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 $r[0], r[L], r[L*2], r[L*3], \dots$ 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 $S[i+L]$  ( $0 \leq i < L$ ) 必定是一样的字符, 而这两个字符位置相差 $L$

而字符 $r[0], r[L], r[L*2], r[L*3], \dots$ 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:

```

79
80 int calc(int i,int j){
81     int x = rk[i];
82     int y = rk[j];
83     if(x>y) swap(x,y);
84     return st.query_min(x+1,y);
85 }
86
87 int ans=1;
88 for(int i=1;i<n;i++){
89     for(int j=1;j+i<=n;j+=i){
90         int lcp = calc(j,j+i);
91         int pos = j-(i-lcp%i);
92         int res = lcp/i+1;
93         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]){

```

```

152         be=p;en=p+repeat*v[j]-1;
153         break;
154     }
155 }
156 }
157 for(;be<=en;be++) printf("%c",s[be]);
158 printf("\n");
159 }
160 }

```

161

162 6) 两个字符串, 求三元组 $(i, j, k)$ , 即 $s[i] \dots s[i+k-1]$ 和 $s[j] \dots s[j+k-1]$ 相同的个数,  $k \geq$  给定 $l$ 。

163 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

164 , 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

165 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

166 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。

167 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

168 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。

```

169
170 int main() {
171     while(scanf("%d",&k)==1){
172         if(!k) break;
173         scanf("%s%s",s+1,s2+1);
174         int len=strlen(s+1);
175         s[0]=1;strcat(s,"#");strcat(s,s2+1);s[0]=0;
176         int Len=strlen(s+1);
177         DA(s,Len,128);
178         ll ans=0,tot=0;
179         int top=0;
180         //a扫b
181         for(int i=1;i<=Len;i++){
182             if(height[i]<k) top=tot=0;
183             else{
184                 ll cnt=0;
185                 if(sa[i-1]<=len) cnt++,tot+=height[i]-k+1;
186                 while(top>0&&st[top-1][0]>=height[i]){
187                     top--;
188                     tot-=st[top][1]*(st[top][0]-height[i]);
189                     cnt+=st[top][1];
190                 }
191                 st[top][0]=height[i];st[top++][1]=cnt;
192                 if(sa[i]>len) ans+=tot;
193             }
194         }
195         //b扫a
196         for(int i=1;i<=Len;i++){
197             if(height[i]<k) top=tot=0;
198             else{
199                 ll cnt=0;
200                 if(sa[i-1]>len) cnt++,tot+=height[i]-k+1;
201                 while(top>0&&st[top-1][0]>=height[i]){
202                     top--;
203                     tot-=st[top][1]*(st[top][0]-height[i]);
204                     cnt+=st[top][1];
205                 }
206                 st[top][0]=height[i];st[top++][1]=cnt;

```

```

207         if(sa[i]<=len) ans+=tot;
208     }
209 }
210 printf("%lld\n",ans);
211 }
212 }
213
214 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
215     只需要将正串和反串都加入后缀数组, 然后计算即可。
216     int t;
217     scanf("%d",&t);
218     while(t--){
219         scanf("%d",&n);
220         int left=1,right=100,len=1,sp=120;
221         for(int i=1;i<=n;i++){
222             scanf("%s",str);
223             int l=strlen(str);
224             right=min(right,l);
225             for(int j=0;j<l;j++){
226                 mark[len] =2*i-1; s[len++] = str[j]-'0'+1;
227             }
228             s[len++]=sp++;
229             for(int j=l-1;j>=0;j--){
230                 mark[len] =2*i; s[len++] = str[j]-'0'+1;
231             }
232             s[len++]=sp++;
233         }
234         len--;
235         DA(s,len,sp++);
236         while(left<=right){
237             int mid=(left+right)>>1;
238             if(isOK(mid,len,sp)) left=mid+1;
239             else right=mid-1;
240         }
241         printf("%d\n",right);
242     }
243
244
245

```

#### 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  $\text{endpos}(p)$

1. 如果两个子串的 $\text{endpos}$ 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 $p$  ( $\text{len}_t \leq \text{len}_p$ ), 要么 $\text{endpos}(p) \supset \text{endpos}(t)$ , 要么 $\text{endpos}(t) \cap \text{endpos}(p) = \emptyset$

3. 对于 $\text{endpos}$ 相同的子串, 我们将它们归为一个 $\text{endpos}$ 等价类。对于任意一个 $\text{endpos}$ 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 $\text{endpos}$ 等价类内的串的长度连续)

4.  $\text{endpos}$ 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 $\text{len}(a)$ , 最短子串长度为 $\text{minlen}(a)$ 。

对于存在父子关系的两个类,

设 $\text{fa}(a)$ 表示类 $a$ 的父亲 (也是一个类)。则:  $\text{len}(\text{fa}(a))+1=\text{minlen}(a)$

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 $\text{endpos}$ 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 $\text{fa}(i)$ 的所有字符串的长度, 且到达  $\text{fa}(i)$ 的任意一字符串必为到达 $i$ 的任

260 意一字符串的后缀。

261 4. 每个节点都代表不同的endpos等价类,  $\text{longest}[i] - \text{minlen}[i] + 1$  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

262  $\text{len}(\text{fa}(a)) + 1 = \text{minlen}(a)$ , 因此每个节点只需要记录最长的  $\text{len}[i]$  即可, 最小值只需要找父节点来确定。

263 5. 数组要开两倍

264

265 学习文章: hihocoder 里面关于后缀自动机讲解

266

267 应用:

268

269 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

270 代码:

```

271 void solve(){
272     ll ans=0;
273     for(int i=1;i<=tot;i++) c[len[i]]++;
274     for(int i=1;i<=tot;i++) c[i]+=c[i-1];
275     for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
276     for(int i=tot;i>=1;i--){
277         int p = rk[i];
278         sz[fa[p]]+=sz[p];
279         if(sz[p]>1)
280             ans=max(ans,(ll)sz[p]*len[p]);
281     }
282     printf("%lld\n",ans);
283 }
284
```

285 2、求一个字符串中长度大于等于m的个数总和

```

286 void solve(){
287     ll ans=0;
288     for(int i=tot;i>=1;i--){
289         if(len[i]<m)
290             continue;
291         else{
292             ans+=(len[i]-max(m,len[fa[i]]+1)+1);
293         }
294     }
295     printf("%lld\n",ans);
296 }
297
```

298 3、求两个字符串的最长公共子串

299 方法1: 建立两个后缀自动机, 然后  $\text{dfs}(1, 1, 0)$ , 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

300 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

```

301 void query(char *s){
302     int ret=0,p=1,nowlen=0;
303     int n=strlen(s);
304     for(int i=0;i<n;i++){
305         int ch=s[i]-'a';
306         if(nx[p][ch]){p=nx[p][ch];ret=max(ret,++nowlen);continue;}
307         //如果没有匹配成功则p=fa[p],保证endpos一样, 有满足 $\text{min}\{p\}=\text{max}\{\text{fa}[p]\}+1$ 
308         //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
309         while(p&&!nx[p][ch]) p=fa[p];
310         if(!p) nowlen=0,p=1;
311         else{
312             //注意不可以 $p=nx[p][c]$ ;  $\text{nowlen}=\text{len}[\text{fa}[p]]+1$ ; 因为fa和nx数组并不是相互映射的关系
313             ret=max(ret,nowlen=len[p]+1); p=nx[p][ch];
314         }
315     }
316     printf("%d\n",ret);

```

317 }  
 318  
 319 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。

320 方法一：先建立对一串建立后缀自动机，然后操作如下：

```

321 inline void topsort(){
322     for(int i=1;i<=tot;i++) c[len[i]]++;
323     for(int i=1;i<=tot;i++) c[i]+=c[i-1];
324     for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
325     for(int i=tot;i>=1;i--){int p=rk[i]; sz[fa[p]]+=sz[p];}
326     //表示状态转移到p节点时，dp[p]能新增多少答案
327     for(int i=1;i<=tot;i++){
328         int p=rk[i];dp[p]=dp[fa[p]]+sz[p]*(len[p]-len[fa[p]]);
329     }
330 }
331
332 inline void solve(char *s){
333     ll ret=0;
334     int p=1,nowlen=0;
335     int n=strlen(s);
336     for(int i=0;i<n;i++){
337         int c=s[i]-'a';
338         if(nx[p][c]){
339             p=nx[p][c]; nowlen++; ret+=dp[fa[p]]+sz[p]*(nowlen-len[fa[p]]);
340             continue;
341         }
342         for(;p&&!nx[p][c];p=fa[p]);
343         if(!p) p=1,nowlen=0;
344         else{
345             nowlen=len[p]+1;
346             p=nx[p][c];
347             ret+=dp[fa[p]]+sz[p]*(nowlen-len[fa[p]]);
348         }
349     }
350     printf("%lld\n",ret);
351 }
  
```

353 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

354 对于i从小到大处理，维护使得 $s[j:i] \sqsupseteq s[1:j-1]$ 的最小的j，那么记f[i]为输出前i个字符的最小代价，则 $f[i]=\min\{f[i-1]+p, f[j-1]+q\}$ 。

355 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

356 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 $s[j, i+1] \sqsupseteq s[1, j-1]$ 。

```

357 class SuffixAutoMaton{
358 public:
359     int last,tot,p;
360     int nx[N<<1][dif],fa[N<<1],len[N<<1];
361     void init(){
362         last=tot=1; p=1;
363         fa[1]=len[1]=0;
364         memset(nx[1],0, sizeof(nx[1]));
365     }
366
367     inline int match(char ch){
368         return nx[p][ch-'a'];
369     }
370
371     inline void withdraw(int l){
  
```

```

372     while(p!=0&&len[fa[p]]>=l) p=fa[p];
373     if(p==0) p=1;
374 }
375
376 void transfer(int l,int ch){
377     p=nx[p][ch];
378     if(p==0) p=1;
379     withdraw(l);
380 }
381
382 inline void insert(int c){
383     int p=last,np=++tot;
384     memset(nx[tot],0, sizeof(nx[tot]));
385     last=np; len[np]=len[p]+1;
386     for(;p&&!nx[p][c];p=fa[p]) nx[p][c]=np;
387     if(!p) fa[np]=1;
388     else{
389         int q=nx[p][c];
390         if(len[p]+1==len[q]) fa[np]=q;
391         else{
392             int nq=++tot;
393             len[nq]=len[p]+1;
394             memcpy(nx[nq],nx[q], sizeof(nx[q]));
395             fa[nq]=fa[q]; fa[q]=fa[np]=nq;
396             for(;nx[p][c]==q;p=fa[p])
397                 nx[p][c]=nq;
398         }
399     }
400 }
401
402 }SAM;
403
404
405 void solve(){
406     SAM.init();
407     SAM.insert(s[0]-'a');
408     dp[0]=p;
409     int l=1,r=0;
410     for(int i=1;i<n;i++){
411         ++r;
412         dp[i]=dp[i-1]+p;
413         while( ( !SAM.match(s[i]) || r-l+1>(i+1)/2 ) && l<=r ){
414             SAM.insert(s[l++]-'a');
415             SAM.withdraw(r-l);
416         }
417         SAM.transfer(r-l+1,s[i]-'a');
418         if(l<=r){
419             dp[i]=min(dp[i],dp[i-(r-l+1)]+q);
420         }
421     }
422     printf("%lld\n",dp[n-1]);
423 }
424

```

425 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”。

428

429 输入：  
 430 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，  
 431 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。

432  
 433 abac  
 434 3  
 435 a  
 436 ab  
 437 ca

438  
 439 输出：  
 440 输出共N行，每行一个整数，表示答案。

441 2  
 442 2  
 443 1

```

444 //将字符串s->s+s,然后进入处理
445 void query(char *s){
446     int n=strlen(s),p=1,nowlen=0,limit=n/2;
447     memset(vis,0, sizeof(bool)*(tot+1));
448     ll ret=0;
449     for(int i=0;i<n-1;i++){
450         int c=s[i]-'a';
451         if(nowlen==limit&&nx[p][c]){
452             nowlen++;p=nx[p][c];
453             if(nowlen>len[fa[p]]+1) nowlen--;
454             else{
455                 nowlen--; p=fa[p];
456             }
457         }else{
458             if(nx[p][c]){
459                 nowlen++;p=nx[p][c];
460             }else {
461                 while (p && !nx[p][c]) p = fa[p];
462                 if (!p) {
463                     p = 1;nowlen = 0;
464                 } else {
465                     nowlen = len[p] + 1;p = nx[p][c];
466                 }
467             }
468         }
469         if(nowlen==limit&&!vis[p]){
470             ret+=sz[p];vis[p]=1;
471         }
472     }
473     printf("%lld\n",ret);
474 }
475
476

```

477 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。  
 478 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。

479 思路：  
 480 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。

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



```

487     for(int i=0;i<26;i++){
488         int v=nx[u][i];
489         if(!v) continue;
490         if(!dp[v]) dfs(v);
491         dp[u]+=dp[v];
492     }
493 }
494
495 //对答案进行计算
496 void DFS(int u,ll k){
497     if(sz[u]>=k) return;
498     k-=sz[u];
499     for(int i=0;i<26;i++){
500         int v=nx[u][i];
501         if(!v) continue;
502         if(dp[v]>=k){
503             printf("%c",i+'a');
504             DFS(v,k); return;
505         }else k-=dp[v];
506     }
507 }
508
509 void query(int t,ll k){
510     topu(t); sz[1]=0; dfs(1);
511     if(dp[1]<k){
512         printf("-1\n");return;
513     }
514     DFS(1,k);
515     printf("\n");
516 }

```

517 8、给出n个串，求这个n个串的最长公共子串， $1 \leq n \leq 10, |s| \leq 1e5$   
518 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mle[n][i]表示i节点，n-1个串匹配长度的最大值中的最小值  
519 那么最后答案就是遍历所有节点的mle，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

```

521
522 代码:
523 bool Update(){
524     if(scanf("%s",s)!=1) return 0;
525     memset(cle,0,sizeof(int)*(tot+1));
526     int n=strlen(s),nowlen=0,p=1;
527     //进行节点匹配, cle[i]表示当前节点匹配的最大值
528     for(int i=0;i<n;i++){
529         int c=s[i]-'a';
530         if(nx[p][c]){
531             p=nx[p][c];nowlen++;
532         }else{
533             while(p&&!nx[p][c]) p=fa[p];
534             if(!p){
535                 p=1;nowlen=0;
536             }else{
537                 nowlen=len[p]+1;p=nx[p][c];
538             }
539         }
540         cle[p]=max(cle[p],nowlen);
541     }
542     //更新mle[i],并更新父节点的cle
543     for(int i=tot;i>=1;i--){

```

```

544     int p=rk[i];
545     mlen[p]=min(mlen[p],clen[p]);
546     if(clen[p]&&fa[p]) clen[fa[p]]=len[fa[p]];
547 }
548 return 1;
549 }
550

```

551 9、给定一个长度为 $n$ 的字符串 $s$ ，令 $T_i$ 表示它从第 $i$ 个字符开始的后缀，求 $\sum_{1 \leq i < j \leq n} \text{len}[i] + \text{len}[j] - 2 * \text{lcp}(T_i, T_j)$ 的和

552 其中， $\text{len}[i]$ 表示字符串 $i$ 的长度， $\text{lcp}(a, b)$ 表示字符串 $a$ 和字符串 $b$ 的最长公共前缀

553

554 思路：

555 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

556 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

557 计算即可。

558

559 代码：

```

560 void query(ll n){
561     ll ret=(n-1)*n*(n+1)/2;
562     for(int i=1;i<=tot;i++) c[len[i]]++;
563     for(int i=1;i<=tot;i++) c[i]+=c[i-1];
564     for(int i=1;i<=tot;i++) rk[c[len[i]]--]=i;
565     for(int i=tot;i>=1;i--){
566         int p=rk[i]; sz[fa[p]]+=sz[p];
567         ret-=(len[p]-len[fa[p]])*sz[p]*(sz[p]-1);
568     }
569     printf("%lld\n",ret);
570 }
571

```

572 广义后缀自动机：

573

574 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：

575 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，

576 允许有前导0）。答案有可能很大，我们需要对 $(10^9 + 7)$ 取模。

577

578 输入：

579 2

580 101

581 09

582

583 输出：

584 131

585

586 思路：

587 建立广义后缀自动机，然后进行拓扑排序，累计答案即可。

588

```

589 ll dp[N<<1],cnt[N<<1];
590 int rk[N<<1],c[N<<1];
591

```

591

```

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

```

```

598     for(int i=1;i<=tot;i++){
599         int u=rk[i];
600         for(int j=0;j<10;j++){
601             int v=nx[u][j];
602             if(!v) continue;
603             cnt[v]=(cnt[v]+cnt[u])%mod;
604             dp[v]=(dp[v]+10*dp[u]%mod+cnt[u]*j%mod)%mod;
605         }
606     }
607     ll ret=0;
608     for(int i=2;i<=tot;i++) ret=(ret+dp[i])%mod;
609     printf("%lld\n",ret);
610 }

```

## 7.9 回文自动机

- 1 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Manacher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，两树之间用fail指针连接。
- 2 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：
  - 3 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但至少有一个字符不同便是本质不同）
  - 4 2. 求串S内每一个本质不同回文串出现的次数
  - 5 3. 求串S内回文串的个数（其实就是1和2结合起来）
  - 6 4. 求以下标i结尾的回文串的个数
- 7 应用：
  - 8 1、求最长回文子串
 

9 回文自动机中节点长点最长的即是答案
  - 10 2、求字符串中本质不同回文子串的数量
 

11 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。
  - 12 3、求两个字符串有多少回文子串能配对的数量
 

13 For example, (1, 3, 1, 3) and (1, 3, 3, 5) are both considered as a valid common

14 palindrome substring

15 between “aba” and “ababa”. aba能匹配两次.

16 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 += 1

17 ll\*pt1.cnt[to1]\*pt2.cnt[to2];
  - 18 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。
 

19 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到右所形成的最长回文串,

20 遍历一遍字符串即可得到答案。
  - 21 5、求一个字符串每个前缀中有多少个本质不同的回文串
 

22 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。
  - 23 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
 

24 数

25 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

26 查询情况。
  - 27 7、定义合法(x,y)为两个不相交的回文串,求个数。
 

28 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

29 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

30 长度即加入一个点后

31 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  $1 \leq i \leq j < k \leq \text{length}(S)$ ,  $S[i..j]$  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){

```

```

90     int to = pt.fail[from];
91     res+=vis[to];
92     res2+=ls[to];
93     res%=mod;
94     res2%=mod;
95 }
96 vis[from]=res;
97 ls[from]=res2;
98 return;
99 }

```

100  
101 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  
102 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

103 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。

```

104
105 typedef pair<int, int> pii;
106 typedef long long ll;
107 const double eps = 1e-6;
108 const ll INF = 0x3f3f3f3f3f3f3f3f;
109 const ll mod = 1000000007;
110 const int N = 2e5 + 10;
111 const int M = 2e5 + 10;
112
113 const double PI = acos(-1.0);
114
115 const int dif = 26;
116 int n, q;
117 int op;
118 char str[5];
119
120
121 class PalindromicTree{
122 public:
123     int nx[N][dif], fail[N], len[N], num[N];
124     int tot[2], p, last[2], s[N];
125     int newnode(int l){
126         memset(nx[p], 0, sizeof(nx[p]));
127         len[p]=l;
128         num[p]=0;
129         return p++;
130     }
131     void init(int x){
132         last[0]=last[1]=p=0;
133         tot[0]=x;tot[1]=x-1;
134         fail[0]=fail[1]=1;
135         memset(s, -1, sizeof(s));
136         newnode(0);
137         newnode(-1);
138     }
139     int getfail(int x,int tag){
140         if(!tag){
141             while(s[tot[tag]+len[x]+1]!=s[tot[tag]])
142                 x=fail[x];
143         }else{
144             while(s[tot[tag]-len[x]-1]!=s[tot[tag]])
145                 x=fail[x];
146         }
147         return x;

```

```

148     }
149     int insert(int x,int tag){
150         if(!tag)
151             s[--tot[0]]=x;
152         else
153             s[++tot[1]]=x;
154         int cur = getfail(last[tag],tag);
155         int now = nx[cur][x];
156         if(!now){
157             now = newnode(len[cur]+2);
158             fail[now]=nx[getfail(fail[cur],tag)][x];
159             nx[cur][x]=now;
160             num[now]+=num[fail[now]]+1;
161         }
162         last[tag]=nx[cur][x];
163         if(len[last[tag]]==tot[1]-tot[0]+1)
164             last[tag^1]=last[tag];
165         return num[last[tag]];
166     }
167 }pt;
168
169
170 int main() {
171     while(scanf("%d",&q)==1){
172         pt.init(q);
173         ll ans=0;
174         while(q--){
175             scanf("%d",&op);
176             if(op==1){
177                 scanf("%s",str);
178                 ans+=pt.insert(str[0]-'a',0);
179             }else if(op==2){
180                 scanf("%s",str);
181                 ans+=pt.insert(str[0]-'a',1);
182             }else if(op==3){
183                 printf("%d\n",pt.p-2);
184             }else{
185                 printf("%lld\n",ans);
186             }
187         }
188     }
189     return 0;
190 }
191
192 10、两个相交的回文串为一对,求一个字符串中有多少对。
193     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
194     内存有限定,因此可以使用
195     vector进行优化。
196     class PalindromicTree{
197     public:
198         vector<pii>nx[N];
199         int fail[N],len[N],num[N];
200         int tot,p,last,s[N];
201         int newnode(int l){
202             nx[p].clear();
203             len[p]=l;
204             num[p]=0;
205             return p++;

```

```

206 void init(){
207     tot=p=last=0;
208     s[0]=-1,fail[0]=1;
209     newnode(0);
210     newnode(-1);
211 }
212 int getfail(int x){
213     while(s[tot-len[x]-1]!=s[tot])
214         x=fail[x];
215     return x;
216 }
217
218 int is_exist(int p,int c){
219     for(auto t:nx[p]){
220         if(t.first==c)
221             return t.second;
222     }
223     return 0;
224 }
225
226 void insert(int x){
227     s[++tot]=x;
228     int cur = getfail(last);
229     int now = is_exist(cur,x);
230     if(!now){
231         now = newnode(len[cur]+2);
232         fail[now]=is_exist(getfail(fail[cur]),x);
233         nx[cur].push_back(make_pair(x,now));
234         num[now]=num[fail[now]]+1;
235     }
236     last=now;
237 }
238 }pt;
239
240 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
241 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
242 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。
243 const ull hash1 = 201326611;
244 const ull hash2 = 50331653;
245 ull ha[N],pp[N];
246
247 ull getha(int l,int r){
248     if(l==0)
249         return ha[r];
250     return ha[r]-ha[l-1]*pp[r-l+1];
251 }
252
253 bool check(int l,int r){
254     int len = r-l+1;
255     int mid = (l+r)>>1;
256     if(len&1)
257         return getha(l,mid)==getha(mid,r);
258     else
259         return getha(l,mid)==getha(mid+1,r);
260 }
261
262 class PalindromicTree{
263 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         }
300         for(int i=2; i<p; i++){
301             if(check(id[i]-len[i], id[i]-1)){
302                 ans[len[i]]+=cnt[i];
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         }
312         while(scanf("%s", str)==1){
313             int len = strlen(str);
314             memset(ans, 0, sizeof(int)*(len+2));
315             pt.init();
316             ha[0]=str[0];
317             for(int i=0; i<len; i++){
318                 pt.insert(str[i]-'a');
319             }
320             for(int i=1; i<len; i++){
321                 ha[i]=ha[i-1]*hash1+str[i];
322             }
323             pt.makecnt();
324             printf("%d", ans[1]);
325             for(int i=2; i<=len; i++){

```



```

323         printf(" %d",ans[i]);
324     printf("\n");
325     }
326 }
327
328 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
329 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.
330 void dfs(int x,ll res){
331     vector<int>v;
332     for(int i=x;i>1;i=fail[i]){
333         if(!vis[i]){
334             v.push_back(i);
335             vis[i]=1;
336             res++;
337         }else break;
338     }
339     ans+=res;
340     for(int i=0;i<26;i++) if(nx[x][i]) dfs(nx[x][i],res);
341     for(auto t:v) vis[t]=0;
342 }
343
344 void solve(){
345     ans=2-p;
346     dfs(0,0);
347     dfs(1,0);
348     printf("%lld\n",ans);
349 }

```

## 7.10 线段树维护 dp

- 1 1.Wi-Fi(<http://codeforces.com/contest/1216/problem/F>)
- 2 题意:
- 3 现在有 $n(1 \leq n \leq 2e5)$ 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 $[\max(1, i-k), \min(n, i+k)]$
- 4 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 $1 \sim n$ .现在求使得 $n$ 个房间全部联通的最小花费
- 5
- 6 思路:
- 7 设 $dp[i]$ 表示 $1 \sim i$ 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.
- 8