

Template

ronaflx@Glare

November 3, 2011

Contents

1	string	3
1.1	ACAutoMachine	3
1.2	KMP	4
1.3	ELFhash	4
1.4	Minimum Representation	5
1.5	C++replaceAll	6
1.6	Base64	6
1.7	Trie Tree	7
1.8	Suffix Array	8
2	data structure	10
2.1	AVL	10
2.2	Splay	12
2.3	LCA(RMQ)	16
2.4	leftist Tree	17
2.5	partition Tree	18
2.6	tree Representation	19
2.7	UFS	19
2.8	BIT_findK	20
2.9	UFS_value	20
2.10	3D.BIT	21
2.11	DLX(exact)	22
2.12	DLX(indistinct)	24
3	graph	27
3.1	BBC	27
3.2	EBBC	28
3.3	scc	29
3.4	2-sat	30
3.5	Hopcroft-Karp	32
3.6	hungary	33
3.7	KM	34
3.8	stableMarriage	35
3.9	maximal Clique	35
3.10	MaxClique	37
3.11	minimum Cut	37
3.12	LCA	38
3.13	bellman-ford	39
3.14	Eular-fleury	40
3.15	k-shortest-path(with cycle)	41
3.16	secondSP	42
3.17	SPFA	44
3.18	Best Radio Spanning Tree	45
3.19	Count Spanning Tree	47
3.20	Degree Limited Spanning Tree	47
3.21	second MST	49
3.22	minimum direct tree(matrix)	51
3.23	minimum direct tree(pool)	53
3.24	maxflow	54
3.25	mincostflow	56
4	computational geometry	58
4.1	geometry	58
4.2	3D-Convex	61

5	math	64
5.1	cantor extend	64
5.2	eular function	64
5.3	Matrix	65
5.4	miller rabin	66
5.5	pollard rho	67
5.6	linearModularSystem	68
5.7	Eratosthenes	69
6	dynamic Programming	70
6.1	DFA DP	70
6.2	mask & connection	70
6.3	RMQ	73
6.4	HOJ2973	73
6.5	slope	74
7	other	76
7.1	连通性DP	76
7.2	input stream	79
7.3	splay	80
7.4	network with low bound	84

1 string

1.1 ACAutoMachine

```
1 #define code(ch) ((ch) - 'a')
2 const int KIND = 26;
3 const int MAXN = 500000;
4 struct node
5 {
6     node* next[KIND], *fail;
7     int count;
8 } pool[MAXN], *pp, *root, *q[MAXN];
9 int head, tail;
10 node *newNode()
11 {
12     pp->fail = NULL;
13     pp->count = 0;
14     memset(pp->next, 0, sizeof (pp->next));
15     return pp++;
16 }
17 void initialize()
18 {
19     pp = pool;
20     root = newNode();
21 }
22 void insert(char *str)
23 {
24     node *now = root;
25     while (*str)
26     {
27         int idx = code(*str);
28         if (now->next[idx] == NULL)
29             now->next[idx] = newNode();
30         now = now->next[idx];
31         str++;
32     }
33     now->count++;
34 }
35 void buildFail(node*& now, int ith) //build now's ith son
36 {
37     if(now == root) now->next[ith]->fail = root;
38     node* tmp = now->fail;
39     while(tmp)
40     {
41         if(tmp->next[ith] != NULL)
42         {
43             now->next[ith]->fail = tmp->next[ith];
44             return;
45         }
46         tmp = tmp->fail;
47     }
48     if(tmp == NULL) now->next[ith]->fail = root;
49 }
50 void build()
51 {
52     head = tail = 0;
53     q[tail++] = root;
54     while (head != tail)
55     {
56         node *beg = q[head++];
57         for (int i = 0; i < KIND; i++)
58         {
59             if (beg->next[i] == NULL) continue;
60             buildFail(beg, i);
61             q[tail++] = beg->next[i];
```

```

62     }
63 }
64 }
65 node* goStatus(node* now, int ith)
66 {
67     while(now->next[ith] == NULL && now != root)
68         now = now->fail;
69     now = now->next[ith];
70     return now == NULL ? root : now;
71 }
72 int query(char* str)
73 {
74     int cnt = 0;
75     node* p = root, *tmp;
76     while (*str)
77     {
78         tmp = p = goStatus(p, code(*str));
79         while (tmp != root && tmp->count != -1)
80         {
81             cnt += tmp->count;
82             tmp->count = -1;
83             tmp = tmp->fail;
84         }
85         str++;
86     }
87     return cnt;
88 }

```

1.2 KMP

```

1  //be careful with mod string and main string
2  void prefix(char *mode, int *next)
3  {
4      int m = strlen(mode), k = -1, i;
5      next[0] = -1;
6      for (i = 1; i < m; i++)
7      {
8          while (k > -1 && mode[k + 1] != mode[i]) k = next[k];
9          if (mode[k + 1] == mode[i]) k++;
10         next[i] = k;
11     }
12 }
13 int KMP(char *main, char *mode)
14 {
15     int n = strlen(main), m = strlen(mode), q = -1, ans = 0;
16     int next[LEN], i;
17     prefix(mode, next);
18     for (i = 0; i < n; i++)
19     {
20         while (q > -1 && mode[q + 1] != main[i]) q = next[q];
21         if (mode[q + 1] == main[i]) q++;
22         if (q == m - 1)
23         {
24             ans++;
25             q = next[q];
26         }
27     }
28     return ans;
29 }

```

1.3 ELFhash

```

1  class hash_table
2  {
3  public:
4      unsigned int ELFhash(char *key)
5      {
6          unsigned int h = 0, g;
7          while (*key)
8          {
9              h = (h << 4) + *key++;
10             g = h & 0xf0000000L;
11             if (g) h ^= g >> 24;
12             h &= ~g;
13         }
14         return h % MOD;
15     }
16     int find(char * str, int judge = 0)
17     {
18         int t = ELFhash(str);
19         for (hashCell * i = g[t]; i != NULL; i = i->next)
20             if (!strcmp(i->str, str))
21                 return i->p = judge ? i->p + 1: i->p;
22         return 0;
23     }
24
25     void insert(char* str, int p)
26     {
27         if(find(str, 1)) return;
28         unsigned t = ELFhash(str);
29         strcpy(pp->str, str);
30         pp->p = p;
31         pp->next = g[t];
32         g[t] = pp++;
33     }
34 private:
35     const static int MOD = 387173;
36     const static int SIZE = 380001;
37     struct hashCell
38     {
39         char str[20];
40         int p;
41         hashCell * next;
42     } pool[SIZE], *g[MOD], *pp;
43 };

```

1.4 Minimum Representation

```

1  int Minimum_Representation(char *s, int len)
2  {
3      int i = 0, j = 1, k = 0, t;
4      while(i < len && j < len && k < len)
5      {
6          t = s[(i + k) % len] - s[(j + k) % len];
7          if(t == 0) k++;
8          else
9          {
10             if(t < 0) j += k + 1;
11             else i += k + 1;
12             if(i == j) j++;
13             k = 0;
14         }
15     }
16     return min(i, j);
17 }

```

1.5 C++replaceAll

```
1 string& replace_all(string& str, const string& src, const string& dest)
2 {
3     string::size_type pos(0);
4     while((pos = str.find(src)) != string::npos)
5         str.replace(pos,src.length(), dest);
6     return str;
7 }
8 string& replace_all_distinct(string& str, const string& src, const string& dest
9 )
10 {
11     for(string::size_type pos(0);(pos = str.find(src, pos)) != string::npos;pos
12         += dest.length())
13         str.replace(pos, src.length(), dest);
14     return str;
15 }
```

1.6 Base64

```
1 static const std::string base64_chars =
2     "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3     "abcdefghijklmnopqrstuvwxyz"
4     "0123456789+/";
5
6
7 static inline bool is_base64(unsigned char c) {
8     return (isalnum(c) || (c == '+') || (c == '/'));
9 }
10
11 std::string base64_encode(unsigned char const* bytes_to_encode, unsigned int
12     in_len) {
13     std::string ret;
14     int i = 0;
15     int j = 0;
16     unsigned char char_array_3[3];
17     unsigned char char_array_4[4];
18
19     while (in_len--) {
20         char_array_3[i++] = *(bytes_to_encode++);
21         if (i == 3) {
22             char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
23             char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0
24                 xf0) >> 4);
25             char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0
26                 xc0) >> 6);
27             char_array_4[3] = char_array_3[2] & 0x3f;
28
29             for(i = 0; (i < 4) ; i++)
30                 ret += base64_chars[char_array_4[i]];
31             i = 0;
32         }
33     }
34
35     if (i)
36     {
37         for(j = i; j < 3; j++)
38             char_array_3[j] = '\0';
39
40         char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
41         char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0
42             ) >> 4);
43         char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0
44             ) >> 6);
```

```

40     char_array_4[3] = char_array_3[2] & 0x3f;
41
42     for (j = 0; (j < i + 1); j++)
43         ret += base64_chars[char_array_4[j]];
44
45     while((i++ < 3))
46         ret += '=';
47
48 }
49
50 return ret;
51
52 }
53
54 std::string base64_decode(std::string const& encoded_string) {
55     int in_len = encoded_string.size();
56     int i = 0;
57     int j = 0;
58     int in_ = 0;
59     unsigned char char_array_4[4], char_array_3[3];
60     std::string ret;
61
62     while (in_len-- && ( encoded_string[in_] != '=') && is_base64(encoded_string[
        in_])) {
63         char_array_4[i++] = encoded_string[in_]; in_++;
64         if (i ==4) {
65             for (i = 0; i <4; i++)
66                 char_array_4[i] = base64_chars.find(char_array_4[i]);
67
68             char_array_3[0] = (char_array_4[0] << 2) + ((char_array_4[1] & 0x30) >> 4)
                ;
69             char_array_3[1] = ((char_array_4[1] & 0xf) << 4) + ((char_array_4[2] & 0
                x3c) >> 2);
70             char_array_3[2] = ((char_array_4[2] & 0x3) << 6) + char_array_4[3];
71
72             for (i = 0; (i < 3); i++)
73                 ret += char_array_3[i];
74             i = 0;
75         }
76     }
77
78     if (i) {
79         for (j = i; j <4; j++)
80             char_array_4[j] = 0;
81
82         for (j = 0; j <4; j++)
83             char_array_4[j] = base64_chars.find(char_array_4[j]);
84
85         char_array_3[0] = (char_array_4[0] << 2) + ((char_array_4[1] & 0x30) >> 4);
86         char_array_3[1] = ((char_array_4[1] & 0xf) << 4) + ((char_array_4[2] & 0x3c)
            >> 2);
87         char_array_3[2] = ((char_array_4[2] & 0x3) << 6) + char_array_4[3];
88
89         for (j = 0; (j < i - 1); j++) ret += char_array_3[j];
90     }
91
92     return ret;
93 }

```

1.7 Trie Tree

```

1 //trie
2 const int SIZE = 33 * 100000;
3 const int KIND = 26;

```



```

4 struct node
5 {
6     node *child[KIND];
7     int final;
8 } pool[SIZE], *root, *last;
9 void build()
10 {
11     last = root = pool;
12     memset(pool, 0, sizeof(pool));
13 }
14 void insert(char *from)
15 {
16     node *p = root;
17     for (char *i = from; *i; i++)
18     {
19         if (p->child[*i - 'a'] == NULL)
20             p->child[*i - 'a'] = ++last;
21         p = p->child[*i - 'a'];
22         p->final++;
23     }
24 }
25 int query(char *from)
26 {
27     node *p = root;
28     for (char *i = from; *i; i++)
29     {
30         p = p->child[*i - 'a'];
31         if (p == NULL) return 0;
32     }
33     return p->final;
34 }

```

1.8 Suffix Array

```

1 const int MAXN = 21000;
2 struct Sfx
3 {
4     int i;
5     int key[2];
6     bool operator < (const Sfx& s) const
7     {
8         return key[0] == s.key[0] ? key[1] < s.key[1] : key[0] < s.key[0];
9     }
10 } sfx[MAXN], temp[MAXN];
11 int rank[MAXN], bucket[MAXN], height[MAXN]; // rank from 0 to n - 1
12 // 精密radixSortecond then first
13 void radixSort(Sfx* in, int n, int idx, Sfx* out)
14 {
15     memset(bucket, 0, sizeof(int) * (n + 1));
16     for (int i = 0; i < n; i++)
17         bucket[in[i].key[idx]]++;
18     for (int i = 1; i <= n; i++)
19         bucket[i] += bucket[i - 1];
20     for (int i = n - 1; i >= 0; i--) // for down
21         out[--bucket[in[i].key[idx]]] = in[i];
22 }
23 void buildSA(const char* text, int n)
24 {
25     for (int i = 0; i < n; i++)
26     {
27         sfx[i].i = sfx[i].key[1] = i;
28         sfx[i].key[0] = text[i];
29     }
30     sort(sfx, sfx + n);

```

```

31     for (int i = 0; i < n; i++)
32         sfx[i].key[1] = 0;
33     int wid = 1;
34     while (wid < n)
35     {
36         rank[sfx[0].i] = 0;
37         for (int i = 1; i < n; i++)
38             rank[sfx[i].i] = rank[sfx[i - 1].i] + (sfx[i - 1] < sfx[i]);
39         for (int i = 0; i < n; i++)
40         {
41             sfx[i].i = i;
42             sfx[i].key[0] = rank[i];
43             sfx[i].key[1] = i + wid < n ? rank[i + wid]: 0;
44         }
45         radixSort(sfx, n, 1, temp);
46         radixSort(temp, n, 0, sfx);
47         wid <= 1;
48     }
49 }
50 void calHeight(const char* text, int* rank, int n)
51 {
52     //h[i] = height[rank[i]], h[i] >= h[i - 1] - 1;
53     for(int i = 0; i < n; i++)
54         rank[sfx[i].i] = i;
55     for(int i = 0, k = 0, j; i < n; i++)
56     {
57         if (rank[i] == 0)
58             height[rank[i]] = 0;
59         else
60         {
61             if(k > 0) k-- ;
62             for (j = sfx[rank[i] - 1].i; text[i + k] == text[j + k]; k++);
63             height[rank[i]] = k;
64         }
65     }
66 }
67 int RMQ[MAXN][20];
68 void buildRMQ(int n, int* height)
69 {
70     for(int i = 1; i <= n; i++) RMQ[i][0] = height[i - 1];
71     for (int j = 1; j <= log(n + 0.00) / log(2.0); j++)
72         for (int i = 1; i + (1 << j) - 1 <= n; i++)
73             RMQ[i][j] = min(RMQ[i][j - 1], RMQ[i + (1 << (j - 1))][j - 1]);
74 }
75 int queryRMQ(int a, int b)
76 {
77     int len = log(b - a + 1.0) / log(2.0);
78     return min(RMQ[a][len], RMQ[b - (1 << len) + 1][len]);
79 }
80 int queryLCP(int a, int b)
81 {
82     a = rank[a] + 1;
83     b = rank[b] + 1;
84     if(a > b) swap(a, b);
85     return queryRMQ(a + 1, b);
86 }

```

2 data structure

2.1 AVL

```
1  const int INF = 100000000;
2  template<typename T>
3  class AVL
4  {
5  public:
6      AVL()
7      {
8          pp = pool;
9          TMP = node(0, 0, NULL, NULL);
10         MYNULL = &TMP;
11         roof = MYNULL;
12     }
13     void clear()
14     {
15         roof = MYNULL;
16         pp = pool;
17     }
18     int size(){return roof->size;}
19     void insert(T k){insert(roof, k);}
20     void erase(T k){erase(roof, k);}
21     bool empty(){return roof == MYNULL;}
22     int findK(int k)
23     {
24         if (k <= 0) return -INF;
25         return findK(roof, k);
26     }
27 private:
28     #define max(a,b) ((a) < (b) ? (b) : (a))
29     struct node
30     {
31         node *lchild, *rchild;
32         T value;
33         int h, size;
34         node() {}
35         node(int h, int size, node * lchild, node * rchild)
36         {
37             this->size = size, this->h = h;
38             this->lchild = lchild, this->rchild = rchild;
39         }
40     };
41     node* roof;
42     static const int N = 100000;
43     node* MYNULL, TMP, pool[N], *pp;
44     int findK(node* &R, int k)
45     {
46         if (k == R->lchild->size + 1)
47             return R->value;
48         else if (k <= R->lchild->size)
49             return findK(R->lchild, k);
50         else if (k > R->size - R->rchild->size)
51             return findK(R->rchild, k + R->rchild->size - R->size);
52     }
53     void fix(node* &R)
54     {
55         R->h = max(R->rchild->h, R->lchild->h) + 1;
56         R->size = R->rchild->size + R->lchild->size + 1;
57     }
58     void rightsinglerotate(node* &R)
59     {
60         node * lc = R->lchild;
61         R->lchild = lc->rchild;
```

```

62     fix(R);
63     lc->rchild = R;
64     R = lc;
65     fix(R);
66 }
67 void leftsingerotate(node* &R)
68 {
69     node * rc = R->rchild;
70     R->rchild = rc->lchild;
71     fix(R);
72     rc->lchild = R;
73     R = rc;
74     fix(R);
75 }
76 void leftdoublerotate(node* &R)
77 {
78     rightsingerotate(R->rchild);
79     leftsingerotate(R);
80 }
81 void rightdoublerotate(node* &R)
82 {
83     leftsingerotate(R->lchild);
84     rightsingerotate(R);
85 }
86 void maintain(node* &R)
87 {
88     if (R->lchild != MYNULL)
89     {
90         if (R->lchild->lchild->h == R->rchild->h + 1)
91             rightsingerotate(R);
92         else if (R->lchild->rchild->h == R->rchild->h + 1)
93             rightdoublerotate(R);
94     }
95     if (R->rchild != MYNULL)
96     {
97         if (R->rchild->rchild->h == R->lchild->h + 1)
98             leftsingerotate(R);
99         else if (R->rchild->lchild->h == R->lchild->h + 1)
100             leftdoublerotate(R);
101     }
102 }
103 void insert(node* &R, T value)
104 {
105     if (R == MYNULL)
106     {
107         R = mynew(value);
108         return;
109     }
110     else if (value <= R->value)
111         insert(R->lchild, value);
112     else if (value > R->value)
113         insert(R->rchild, value);
114     fix(R);
115     maintain(R);
116 }
117 void erase(node* &R, T value)
118 {
119     if (R == MYNULL)
120         return;
121     if (R->value == value)
122     {
123         if (R->rchild == MYNULL)
124         {
125             node * tmp = R;
126             R = tmp->lchild;
127         }

```

```

128         else
129         {
130             node *tmp = R->rchild;
131             while (tmp->lchild != MYNULL)
132                 tmp = tmp->lchild;
133             R->value = tmp->value;
134             erase(R->rchild, tmp->value);
135             fix(R);
136         }
137         return;
138     }
139     else if (value < R->value)
140         erase(R->lchild, value);
141     else if (value > R->value)
142         erase(R->rchild, value);
143     fix(R);
144     maintain(R);
145 }
146 node* mynew(T value)
147 {
148     pp->lchild = pp->rchild = MYNULL;
149     pp->size = pp->h = 1;
150     pp->value = value;
151     return pp++;
152 }
153 #undef max
154 };

```

2.2 Splay

```

1  //be careful with pushDown
2  template<typename T = int>
3  class Splay
4  {
5  private:
6      #define SIZE(x) ((x) ? (x)->size : 0)
7      #define SUM(x) ((x) ? (x)->sum : 0)
8      #define VAL(x) ((x) ? (x)->val : 0)
9      #define CENTRE (root->ch[1]->ch[0])
10     struct node
11     {
12         int size, sum, add;
13         node* ch[2], *pre;
14         T v;
15         node(T v = T(), int size = 1, node* l = NULL,
16             node* r = NULL, node* pre = NULL)
17         {
18             this->v = v;
19             sum = add = 0;
20             this->size = size;
21             this->pre = pre;
22             ch[0] = l, ch[1] = r;
23         }
24     };
25     node * root;
26     void pushDown(node*& x)
27     {
28         // node* y = x->ch[0];
29         // if(y)
30         // {
31         //     y->add += x->add;
32         //     y->sum += SIZE(y) * x->add;
33         // }
34         // y = x->ch[1];

```

```

35 // if(y)
36 // {
37 //   y->add += x->add;
38 //   y->sum += SIZE(y) * x->add;
39 // }
40 // x->add = 0;
41 }
42
43 void pushUp(node*& x)
44 {
45     x->size = SIZE(x->ch[0]) + SIZE(x->ch[1]) + 1;
46     x->sum = SUM(x->ch[0]) + SUM(x->ch[1]) + x->add;
47 }
48 void rotate(node* x, int type)
49 {
50     node *y = x->pre;
51     pushDown(y);
52     pushDown(x);
53     y->ch[!type] = x->ch[type];
54     if (x->ch[type] != NULL) x->ch[type]->pre = y;
55     x->pre = y->pre;
56     if (y->pre != NULL)
57     {
58         if (y->pre->ch[0] == y) y->pre->ch[0] = x;
59         else y->pre->ch[1] = x;
60     }
61     x->ch[type] = y, y->pre = x;
62     if (y == root) root = x;
63     pushUp(y);
64     pushUp(x);
65 }
66 void insert(node* &R, T v = T(), node* f = NULL)
67 {
68     if (R == NULL)
69     {
70         R = new node(v, 1, NULL, NULL, f);
71         splay(R, NULL);
72         return;
73     }
74     else if (v <= R->v) insert(R->ch[0], v, R);
75     else if (v > R->v) insert(R->ch[1], v, R);
76 }
77 void clear(node*& root)
78 {
79     if(root == NULL) return;
80     clear(root->ch[0]);
81     clear(root->ch[1]);
82     delete root;
83     root = NULL;
84 }
85 node* join(node*& x, node*& y)
86 {
87     if(x == NULL) return y;
88     if(y == NULL) return x;
89     x->pre = y->pre = NULL;
90     node* z = x;
91     while(z->ch[1] != NULL) z = z->ch[1];
92     splay(z, NULL);
93     z->ch[1] = y;
94     y->pre = z;
95     pushDown(z);
96     return z;
97 }
98 void splay(node* x, node* f)
99 {
100     pushDown(x);

```

```

101     while(x->pre != f)
102     {
103         if (x->pre->pre == f)
104         {
105             if (x->pre->ch[0] == x) rotate(x, 1);
106             else rotate(x, 0);
107         }
108         else
109         {
110             node *y = x->pre, *z = y->pre;
111             if (z->ch[0] == y)
112             {
113                 if (y->ch[0] == x) // 1
114                     rotate(y, 1), rotate(x, 1);
115                 else // z
116                     rotate(x, 0), rotate(x, 1);
117             }
118             else
119             {
120                 if (y->ch[1] == x) // 1
121                     rotate(y, 0), rotate(x, 0);
122                 else // z
123                     rotate(x, 1), rotate(x, 0);
124             }
125         }
126     }
127     pushUp(x);
128 }
129 node* MaxOrMin(node* x, int type)//0 minimum 1 maximum
130 {
131     if(x == NULL) return x;
132     while(x->ch[type] != 0) x = x->ch[type];
133     return x;
134 }
135 public:
136 Splay(): root(NULL) {}
137 void insert(T v)
138 {
139     insert(root, v);
140 }
141 int size()
142 {
143     return SIZE(root);
144 }
145 void clear()
146 {
147     clear(root);
148 }
149 node* MaxOrMin(int type)
150 {
151     return MaxOrMin(root, type);
152 }
153 int rank(T v)
154 {
155     node* x = find(root, v);
156     return SIZE(x) + 1;
157 }
158 node* selectK(int k)
159 {
160     node* x = root;
161     while(x != NULL && SIZE(x->ch[0]) + 1 != k)
162     {
163         if(k <= SIZE(x->ch[0])) x = x->ch[0];
164         else
165         {
166             k -= SIZE(x->ch[0]) + 1;

```

```

167         x = x->ch[1];
168     }
169 }
170 if(x != NULL) splay(x, NULL);
171 return x;
172 }
173 node* find(T v)
174 {
175     node* x = root;
176     while(x != NULL && x->v != v)
177         x = x->ch[v > x->v];
178     return x;
179 }
180 void erase(T v)
181 {
182     node* x = find(v);
183     if(x == NULL) return;
184     splay(x, NULL);
185     root = join(x->ch[0], x->ch[1]);
186     if(root != NULL)
187         root->pre = NULL;
188     delete x;
189 }
190 node* PreOrSuc(T v, int type)// 0 predecessor 1 successor
191 {
192     node* x = find(v);
193     if(x == NULL) return NULL;
194     else return MaxOrMin(x->ch[type], !type);
195 }
196 void update(int l, int r, int c)
197 {
198     node* ll = find(l - 1);
199     splay(ll, NULL);
200     node* rr = find(r + 1);
201     splay(rr, root);
202     if(root->ch[1] == NULL || CENTRE == NULL) return;
203     CENTRE->add += c;
204     CENTRE->sum += c;
205 }
206 int query(int l, int r)
207 {
208     node* ll = find(l - 1);
209     splay(ll, NULL);
210     node* rr = find(r + 1);
211     splay(rr, root);
212     if(root->ch[1] == NULL || CENTRE == NULL) 0;
213     return CENTRE->sum;
214 }
215 };
216 Splay<int> spl;
217 int main()
218 {
219     int n, t;
220     int a, b, cases = 1;
221     char cmd[100];
222     scanf("%d", &t);
223     while(t-- && scanf("%d", &n) == 1)
224     {
225         spl.clear();
226         for(int i = 0; i <= n + 1; i++) spl.insert(i);
227         for(int i = 1; i <= n; i++)
228         {
229             scanf("%d", &a);
230             spl.update(i, i, a);
231         }
232         printf("Case %d:\n", cases++);

```



```

233     while (scanf("%s", cmd) == 1 && strcmp(cmd, "End"))
234     {
235         scanf("%d %d", &a, &b);
236         if (!strcmp(cmd, "Query"))
237         {
238             printf("%d\n", spl.query(a, b));
239         }
240         else if (!strcmp(cmd, "Add"))
241         {
242             spl.update(a, a, b);
243         }
244         else
245         {
246             spl.update(a, a, -b);
247         }
248     }
249 }
250 return 0;
251 }

```

2.3 LCA(RMQ)

```

1  const int SIZE = 10000;
2  int color[SIZE], interval[SIZE * 2 + 1], col;
3  int first[SIZE], depth[SIZE], uncolor[SIZE];
4  //color records the color of every node
5  vector<int> adj[SIZE]; bool check[SIZE];
6  int n, pt, q, tim, dp[SIZE * 2 + 1][15];
7  void dfs(int x, int d)
8  {
9      check[x] = true;
10     interval[pt++] = col;
11     first[col] = pt - 1;
12     color[x] = col;
13     uncolor[col++] = x;
14     depth[x] = d;
15     int s = adj[x].size(), w;
16     for (int i = 0; i < s; i++)
17     {
18         w = adj[x][i];
19         if (!check[w])
20         {
21             dfs(w, d + 1);
22             interval[pt++] = color[x];
23         }
24     }
25 }
26 void initialize()
27 {
28     scanf("%d %d", &n, &q);
29     int a, b;
30     for (int i = 0; i < n; i++)
31         adj[i].clear();
32     for (int i = 1; i < n; i++)
33     {
34         scanf("%d %d", &a, &b);
35         adj[a].push_back(b);
36         adj[b].push_back(a);
37     }
38     tim = pt = col = 0;
39     memset(check, false, sizeof (check));
40 }
41 int main()
42 {

```

```

43     int t;
44     scanf("%d", &t);
45     while (t--)
46     {
47         initialize();
48         dfs(0, 1);
49         for (int i = 0; i < pt; i++)
50             dp[i + 1][0] = interval[i];
51         for (int j = 1; j <= log(pt) / log(2); j++)
52         {
53             for (int i = 1; i + (1 << j) - 1 <= pt; i++)
54             {
55                 dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
56             }
57         }
58         while (q--)
59         {
60             int a, b, len, ans;
61             scanf("%d %d", &a, &b);
62             int tempa = a, tempb = b;
63             a = first[color[a]] + 1;
64             b = first[color[b]] + 1;
65             if (a > b)
66             {
67                 int temp = a;
68                 a = b;
69                 b = temp;
70             }
71             len = b - a + 1;
72             len = log(len) / log(2) + 0.00001;
73             ans = min(dp[a][len], dp[b - (1 << len) + 1][len]);
74             printf("%d\n", depth[tempa] + depth[tempb] - 2 * depth[uncolor[ans]]);
75         }
76     }
77     return 0;
78 }

```

2.4 leftist Tree

```

1  #define CMP(a, b) ((a) > (b))
2  #define DIST(v) ((v == NULL) ? -1 : (v->dist))
3  //must be careful when clear after merge
4  //because of the pointer could not be NULL
5  //especially when use new just makeNULL when memory is enough
6  template<typename T>
7  class leftist_tree
8  {
9  private:
10     class node
11     {
12     public:
13         T v;
14         int dist;
15         node *rr, *ll;
16         node(){rr = ll = NULL; dist = 0;}
17         node(T v){this->v = v; rr = ll = NULL; dist = 0;}
18     };
19     node* root;
20     int s;
21     node* merge(node* &left, node* &right)
22     {
23         if(left == NULL) return right;
24         if(right == NULL) return left;
25         if(CMP(right->v, left->v)) swap(left, right);

```

```

26     left->rr = merge(left->rr, right);
27     if(DIST(left->rr) > DIST(left->ll)) swap(left->ll, left->rr);
28     left->dist = DIST(left->rr) + 1;
29     return left;
30 }
31 void clear(node* root)
32 {
33     if(root == NULL) return;
34     clear(root->ll);
35     clear(root->rr);
36     delete root;
37     root = NULL;
38 }
39 public:
40 leftist_tree(){root = NULL;s = 0;}
41 ~leftist_tree(){clear(root);}
42 void push(T v)
43 {
44     node * newNode = new node(v);
45     root = merge(newNode, root);
46     s++;
47 }
48 void clear(){clear(root);}
49 int size(){return this->s;}
50 T top(){return root->v;}
51 void pop()
52 {
53     node *tmp = root;
54     root = merge(root->ll, root->rr);
55     delete tmp;
56     s--;
57 }
58 void merge(leftist_tree<T>& tree)
59 {
60     this->root = merge(root, tree.root);
61     s += tree.s;
62     tree.root = NULL;
63 }
64 void makeNULL(){root = NULL;}
65 };

```

2.5 partition Tree

```

1  /* NlogN find Kth number in any interval */
2  class partition_tree
3  {
4  private:
5      static const int N = 100005;
6      static const int DEPTH = 20;
7      int tree[DEPTH][N * 4], sorted[N];
8      int toleft[DEPTH][N * 4];
9      int n;
10 public:
11     void initialize(int n, int *array)
12     {
13         this->n = n;
14         for (int i = 1; i <= n; i++)
15             sorted[i] = tree[0][i] = array[i];
16         sort(sorted + 1, sorted + n + 1);
17     }
18     void build(int l, int r, int depth)
19     {
20         if (l == r) return;
21         int mid = (l + r) / 2, same = 0, less = 0;

```

```

22     for (int i = l; i <= r; i++)
23         less += (tree[depth][i] < sorted[mid]);
24     same = mid - l + 1 - less;
25     int lpos = l, rpos = mid + 1;
26     for (int i = l; i <= r; i++)
27     {
28         int w = tree[depth][i];
29         if (w < sorted[mid]) tree[depth + 1][lpos++] = w;
30         else if (w == sorted[mid] && same)
31         {
32             tree[depth + 1][lpos++] = w;
33             same--;
34         }
35         else
36             tree[depth + 1][rpos++] = w;
37         toleft[depth][i] = toleft[depth][l - 1] + lpos - 1;
38     }
39     build(l, mid, depth + 1);
40     build(mid + 1, r, depth + 1);
41 }
42 // ptree.query(l, n, a, b, 0, k) th kth number of [a, b]
43 int query(int L, int R, int l, int r, int depth, int k)
44 {
45     if (l == r) return tree[depth][l];
46     int cnt, mid, tmp1, tmpr;
47     mid = (R + L) / 2, cnt = toleft[depth][r] - toleft[depth][l - 1];
48     if (cnt >= k)
49     {
50         tmp1 = L + toleft[depth][l - 1] - toleft[depth][L - 1];
51         tmpr = tmp1 + cnt - 1;
52         return query(L, mid, tmp1, tmpr, depth + 1, k);
53     }
54     else
55     {
56         tmpr = r + toleft[depth][R] - toleft[depth][r];
57         tmp1 = tmpr - (r - l - cnt);
58         return query(mid + 1, R, tmp1, tmpr, depth + 1, k - cnt);
59     }
60 }
61 };

```

2.6 tree Representation

2.7 UFS

```

1  int father[N], rank[N];
2  int find_set(int x)
3  {
4      return father[x] = father[x] == x ? x : find_set(father[x]);
5  }
6  bool union_set(int x, int y)
7  {
8      if (x != y)
9      {
10         if (rank[x] < rank[y]) father[x] = y;
11         else
12         {
13             rank[x] += rank[x] == rank[y];
14             father[y] = x;
15         }
16         return false;

```

```

17     }
18     return true;
19 }
20 bool link_set(int x, int y)
21 {
22     return union_set(find_set(x), find_set(y));
23 }
24
25 void initialize(int n)
26 {
27     for(int i = 1; i <= n; i++)
28     {
29         rank[i] = 0;
30         father[i] = i;
31     }
32 }

```

2.8 BIT_findK

```

1  /* make sure that the sum is not lower than k*/
2  int findK(int K)
3  {
4      int ans = 0, cnt = 0;
5      for (int i = log(MAXN - 1) / log(2); i >= 0; i--)
6      {
7          ans += (1 << i);
8          if (ans >= MAXN || cnt + c[ans] >= K)
9              ans -= (1 << i);
10         else
11             cnt += c[ans];
12     }
13     return ans + 1;
14 }

```

2.9 UFS_value

```

1  /* HOJ cube stacking*/
2  #include <iostream>
3  #include <cstring>
4  #include <cstdio>
5  using namespace std;
6  const int SIZE = 300001;
7  int cnt[SIZE], dis[SIZE], father[SIZE];
8
9  void initialize()
10 {
11     for (int i = 0; i < SIZE; i++)
12     {
13         cnt[i] = 1;
14         dis[i] = 0;
15         father[i] = i;
16     }
17 }
18
19 int set_find(int x)
20 {
21     if (x != father[x])
22     {
23         int f = father[x];
24         father[x] = set_find(father[x]);
25         dis[x] += dis[f];
26     }

```

```

27     return father[x];
28 }
29 int main()
30 {
31     int t, x, y;
32     char ch;
33     while (scanf("%d", &t) == 1)
34     {
35         initialize();
36         while (t--)
37         {
38             scanf(" %c", &ch);
39             if (ch == 'M')
40             {
41                 scanf("%d %d", &x, &y);
42                 int f1 = set_find(x), f2 = set_find(y);
43                 if (f1 != f2)
44                 {
45                     dis[f2] = cnt[f1];
46                     father[f2] = f1;
47                     cnt[f1] += cnt[f2];
48                 }
49             }
50             else if (ch == 'C')
51             {
52                 scanf("%d", &x);
53                 int f = set_find(x);
54                 printf("%d\n", cnt[f] - dis[x] - 1);
55             }
56         }
57     }
58     return 0;
59 }

```

2.10 3D_BIT

```

1  /*3D BIT escape index 0*/
2  const int size = 201;
3  int c[size][size][size], n;
4  int lowbit(int k)
5  {
6      return k & (k ^ (k - 1));
7  }
8  int sum(const int x1, const int y1, const int z1)
9  {
10     int s = 0;
11     for(int i = x1; i > 0; i -= lowbit(i))
12         for(int j = y1; j > 0; j -= lowbit(j))
13             for(int k = z1; k > 0; k -= lowbit(k))
14                 s += c[i][j][k];
15     return s;
16 }
17
18 void modify(const int x1, const int y1, const int z1, int val)
19 {
20     for(int i = x1; i < size; i += lowbit(i))
21         for(int j = y1; j < size; j += lowbit(j))
22             for(int k = z1; k < size; k += lowbit(k))
23                 c[i][j][k] += val;
24 }
25 /* scanf("%d %d %d %d %d %d %d", &a, &b, &d, &x, &y, &z, &val);
26 * modify(a, b, d, val); modify(x + 1, b, d, -val);
27 * modify(a, y + 1, d, -val); modify(x + 1, y + 1, d, val);
28 * modify(a, b, z + 1, -val); modify(a, y + 1, z + 1, val);

```

```
29 * modify(x + 1, b, z + 1, val); modify(x + 1, y + 1, z + 1, -val);*/
```

2.11 DLX(exact)

```
1  const int SIZE = 16; //here
2  const int SQRTSIZE = 4; //here
3  const int ALLSIZE = SIZE * SIZE, ROW = SIZE * SIZE * SIZE;
4  const int INF = 100000000, COL = SIZE * SIZE * 4;
5  const int N = ROW * COL, HEAD = 0;
6
7  #define BLOCK(r, c) ((r) * SQRTSIZE + c)
8  #define CROW(r, c, k) ((r) + (c) * SIZE + (k) * SIZE * SIZE)
9  #define ROWCOL(i, j) ((i) * SIZE + (j))
10 #define ROWCOLOR(i, k) (ALLSIZE + (i) * SIZE + k)
11 #define COLCOLOR(j, k) (2 * ALLSIZE + (j) * SIZE + k)
12 #define BLOCKCOLOR(i, j, k) (3*ALLSIZE+BLOCK((i/SQRTSIZE), (j/SQRTSIZE))*SIZE+(k
    ))
13
14 int maps[ROW][COL];
15 int ans[N];
16 char sudoku[SIZE][SIZE];
17 int r[N], l[N], u[N], d[N], c[N], s[N];
18 int n, m, ansd, row[N];
19
20 void resume(const int col)
21 {
22     for (int i = u[col]; i != col; i = u[i])
23     {
24         for (int j = l[i]; j != i; j = l[j])
25         {
26             u[d[j]] = j;
27             d[u[j]] = j;
28             s[c[j]]++;
29         }
30     }
31     r[l[col]] = col;
32     l[r[col]] = col;
33 }
34
35 void cover(const int col)
36 {
37     r[l[col]] = r[col];
38     l[r[col]] = l[col];
39     for (int i = d[col]; i != col; i = d[i])
40     {
41         for (int j = r[i]; j != i; j = r[j])
42         {
43             u[d[j]] = u[j];
44             d[u[j]] = d[j];
45             s[c[j]]--;
46         }
47     }
48 }
49
50 void initialize(int n, int m)
51 {
52     l[HEAD] = m;
53     r[HEAD] = 1;
54     for (int i = 1; i <= m; i++)
55     {
56         if (i == m)
57             r[i] = HEAD;
58         else
59             r[i] = i + 1;
```

```

60     l[i] = i - 1;
61     c[i] = u[i] = d[i] = i;
62     s[i] = 0;
63 }
64 int size = m;
65 for (int i = 1; i <= n; i++)
66 {
67     int first = 0;
68     for (int j = 1; j <= m; j++)
69     {
70         if (maps[i - 1][j - 1] == 0)
71             continue;
72         size++;
73         int tmp = u[j];
74         u[j] = size;
75         d[tmp] = size;
76         d[size] = j;
77         u[size] = tmp;
78         if (!first)
79         {
80             first = size;
81             l[size] = r[size] = size;
82         }
83         else
84         {
85             tmp = l[first];
86             r[tmp] = size;
87             l[size] = tmp;
88             l[first] = size;
89             r[size] = first;
90         }
91         row[size] = i;
92         s[j]++;
93         c[size] = j;
94     }
95 }
96 }
97
98 bool dfs(int depth)
99 {
100     if (r[HEAD] == HEAD)
101     {
102         ansd = depth;
103         return true;
104     }
105     int minn = INF, v;
106     for (int i = r[HEAD]; i != HEAD; i = r[i])
107     {
108         if (s[i] < minn)
109         {
110             v = i;
111             minn = s[i];
112         }
113     }
114     cover(v);
115     for (int i = d[v]; i != v; i = d[i])
116     {
117         for (int j = r[i]; j != i; j = r[j])
118             cover(c[j]);
119         ans[depth] = row[i] - 1;
120         if (dfs(depth + 1))
121             return true;
122         for (int j = l[i]; j != i; j = l[j])
123             resume(c[j]);
124     }
125     resume(v);

```



```

126     ans[depth] = -1;
127     return false;
128 }
129
130 int main()
131 {
132     n = ROW;
133     m = COL;
134     while (scanf(" %c", &sudoku[0][0]) == 1)
135     {
136         for(int i = 0; i < SIZE; i++)
137         {
138             for(int j = 0; j < SIZE; j++)
139             {
140                 if(i + j)
141                     scanf(" %c", &sudoku[i][j]);
142             }
143         }
144         memset(maps, 0, sizeof (maps));
145         for (int i = 0; i < SIZE; i++)
146         {
147             for (int j = 0; j < SIZE; j++)
148             {
149                 if (sudoku[i][j] == '-')
150                 {
151                     for (int k = 0; k < SIZE; k++)
152                     {
153                         maps[CROW(i, j, k)][ROWCOL(i, j)] = 1;
154                         maps[CROW(i, j, k)][ROWCOLOR(i, k)] = 1;
155                         maps[CROW(i, j, k)][COLCOLOR(j, k)] = 1;
156                         maps[CROW(i, j, k)][BLOCKCOLOR(i, j, k)] = 1;
157                     }
158                 }
159                 else
160                 {
161                     int k = sudoku[i][j] - 'A';//here
162                     maps[CROW(i, j, k)][ROWCOL(i, j)] = 1;
163                     maps[CROW(i, j, k)][ROWCOLOR(i, k)] = 1;
164                     maps[CROW(i, j, k)][COLCOLOR(j, k)] = 1;
165                     maps[CROW(i, j, k)][BLOCKCOLOR(i, j, k)] = 1;
166                 }
167             }
168             initialize(n, m);
169         }
170         if (dfs(0))
171         {
172             for (int i = 0; i < ansd; i++)
173                 sudoku[ans[i] % SIZE][ans[i] % ALLSIZE / SIZE] = ans[i] / ALLSIZE +
174                     'A';//here
175             for(int i = 0; i < SIZE; i++)
176             {
177                 for(int j = 0; j < SIZE; j++)
178                     putchar(sudoku[i][j]);
179                 puts("");
180             }
181             puts("");
182         }
183         return 0;
184     }

```

2.12 DLX(indistinct)

```

1 const int ROW = 56;

```

```

2  const int COL = 56;
3  const int N = ROW * COL, HEAD = 0;
4  const int INF = 1000000000;
5  int maps[ROW][COL], ansq[ROW], row[N];
6  int s[COL], u[N], d[N], l[N], r[N], c[N];
7  void build(int n, int m)
8  {
9      r[HEAD] = 1;
10     l[HEAD] = m;
11     for (int i = 1; i <= m; i++)
12     {
13         l[i] = i - 1;
14         r[i] = (i + 1) % (m + 1);
15         c[i] = d[i] = u[i] = i;
16         s[i] = 0;
17     }
18     int size = m;
19     for (int i = 1; i <= n; i++)
20     {
21         int first = 0;
22         for (int j = 1; j <= m; j++)
23         {
24             if (!maps[i - 1][j - 1]) continue;
25             size++;
26             d[u[j]] = size;
27             u[size] = u[j];
28             d[size] = j;
29             u[j] = size;
30             if (!first)
31             {
32                 first = size;
33                 l[size] = size;
34                 r[size] = size;
35             }
36             else
37             {
38                 l[size] = l[first];
39                 r[size] = first;
40                 r[l[first]] = size;
41                 l[first] = size;
42             }
43             c[size] = j;
44             // row[size]=i;
45             s[j]++;
46         }
47     }
48 }
49 inline void coverc(int col)
50 {
51     for(int i = d[col]; i != col; i = d[i])
52     {
53         r[l[i]] = r[i];
54         l[r[i]] = l[i];
55     }
56 }
57 inline void resumec(int col)
58 {
59     for(int i = u[col]; i != col; i = u[i])
60     {
61         l[r[i]] = i;
62         r[l[i]] = i;
63     }
64 }
65 bool vis[COL];
66 int H()
67 {

```

```

68     int cnt = 0;
69     memset(vis,0,sizeof(vis));
70     for (int i = r[HEAD]; i != HEAD; i = r[i])
71     {
72         if (vis[i]) continue;
73         cnt++;
74         vis[i] = 1;
75         for (int j = d[i]; j != i; j = d[j])
76             for (int k = r[j]; k != j; k = r[k])
77                 vis[c[k]] = 1;
78     }
79     return cnt;
80 }
81 int cut,nextcut;
82 bool dfs(int dep)
83 {
84     if (!r[HEAD]) return true;
85     int now, minn = ROW;
86     for (int i = r[HEAD]; i != HEAD; i = r[i])
87         if (minn > s[i])
88         {
89             minn = s[i];
90             now = i;
91         }
92     for (int j = d[now]; j != now; j = d[j])
93     {
94         //ansq[dep]=row[rp];
95         coverc(j);
96         for (int i = r[j]; i != j; i = r[i])
97             coverc(i);
98
99         int tmp = dep + 1 + H();
100        if(tmp > cut)
101            nextcut = min(tmp, nextcut);
102        else if (dfs(dep + 1))
103            return true;
104        for (int i = l[j]; i != j; i = l[i])
105            resumec(i);
106        resumec(j);
107    }
108    return false;
109 }
110 int IDAstar(int n)
111 {
112     cut = H();
113     nextcut = n;
114     memset(vis,0,sizeof(vis));
115     while(!dfs(HEAD))
116     {
117         cut = nextcut;
118         nextcut = n;
119     }
120     return cut;
121 }

```

3 graph

3.1 BBC

```
1  /* hoj 1789 Electricity
2   * the graph is not connected
3   * cnt records the number of BBC, it's an cut P if != 0*/
4  const int V = 10000;
5  vector<int> adj[V];
6  int low[V], dfn[V], cnt[V], depth;
7  void initialize(int n)
8  {
9      REP(i, 0, n) adj[i].clear();
10     CC(cnt, 0); CC(dfn, 0);
11     depth = 0;
12 }
13 void dfs(int x, const int ROOT)
14 {
15     low[x] = dfn[x] = ++depth;
16     int s = adj[x].size(), w, num = 0;
17     REP(i, 0, s)
18     {
19         w = adj[x][i];
20         if (!dfn[w])
21         {
22             num++;
23             dfs(w, ROOT);
24             low[x] = min(low[w], low[x]);
25             if (x == ROOT && num >= 2)
26                 cnt[x]++;
27             if (x != ROOT && dfn[x] <= low[w])
28                 cnt[x]++;
29         }
30         else low[x] = min(low[x], dfn[w]);
31     }
32 }
33 int solve(int n)
34 {
35     int cc = 0;
36     REP(i, 0, n)
37     {
38         if (dfn[i] == 0)
39         {
40             dfs(i, i);
41             cc++;
42         }
43     }
44     return cc;
45 }
46 int main()
47 {
48     int n, m, x, y;
49
50     while (scanf("%d %d", &n, &m) == 2 && n + m)
51     {
52         initialize(n);
53         REP(i, 0, m)
54         {
55             scanf("%d %d", &x, &y);
56             adj[x].push_back(y);
57             adj[y].push_back(x);
58         }
59         int ans = solve(n);
60         if (m == 0) printf("%d\n", n - 1);
61         else printf("%d\n", ans + *max_element(cnt, cnt + n));
```

```

62     }
63     return 0;
64 }

```

3.2 EBBC

```

1  /*HOJ2360
2  * idx is new node of the tree
3  * pool should be big enough */
4  const int SIZE = 5000, ROOT = 0, E = 80000;
5  struct edge
6  {
7      int v, id;
8      edge *nxt;
9  } pool[E], *g[SIZE], *pp, *bg[SIZE];
10 stack<int> st;
11 bool flag[E]; //label the edge in case of multi-edge
12 int depth, ebcc, dfn[SIZE], low[SIZE], idx[SIZE];
13 void initialize()
14 {
15     memset(g, 0, sizeof(g));
16     memset(flag, 0, sizeof(flag));
17     memset(bg, 0, sizeof(bg));
18     memset(dfn, 0, sizeof(dfn));
19     pp = pool, depth = 1, ebcc = 0;
20 }
21 void addedge(int v, int w, edge *g[], int id = 0)
22 {
23     pp->v = w, pp->nxt = g[v];
24     pp->id = id, g[v] = pp++;
25 }
26 void dfs(int v)
27 {
28     st.push(v);
29     dfn[v] = low[v] = depth++;
30     int w, x;
31     for (edge* i = g[v]; i != NULL; i = i->nxt)
32     {
33         w = i->v;
34         if (flag[i->id]) continue;
35         flag[i->id] = true;
36         if (dfn[w]) low[v] = min(low[v], dfn[w]);
37         else
38         {
39             dfs(w);
40             low[v] = min(low[v], low[w]);
41             if (low[w] > dfn[v])
42             {
43                 ebcc++;
44                 do
45                 {
46                     x = st.top();
47                     st.pop();
48                     idx[x] = ebcc;
49                 }while (x != w);
50             }
51         }
52     }
53 }
54 void solve() //find out the cut and build the tree
55 {
56     dfs(ROOT); //ROOT = 0 as usual
57     if (!st.empty()) ebcc++;
58     while (!st.empty())

```

```

59     {
60         idx[st.top()] = ebcc;
61         st.pop();
62     }
63 }

```

3.3 scc

```

1  /* tarjan-scc, new graph is a dag from 0 to sccCnt - 1
2  tms is the topo-order*/
3  const int V = 50001, E = 150000;
4  struct edge//graph
5  {
6      int v;
7      edge *nxt;
8  } pool[E * 3], *g[V], *pp, *gscc[V];
9  int st[V], top, tms[V], pt;//toposort
10 bool reach[V];//reach is used to label is reached or not
11 int dfn[V], low[V], idx[V], sccCnt, depth;
12 void addedge(int u, int v, edge* g[])
13 {
14     pp->v = v;
15     pp->nxt = g[u];
16     g[u] = pp++;
17 }
18 void initialize(int n)
19 {
20     memset(g, 0, sizeof (g));
21     memset(reach, false, sizeof (reach));
22     memset(dfn, 0, sizeof (dfn));
23     pp = pool, depth = pt = top = sccCnt = 0;
24 }
25 void dfs(int x)
26 {
27     st[++top] = x;
28     dfn[x] = low[x] = ++depth;
29     int w;
30     for (edge *i = g[x]; i != NULL; i = i->nxt)
31     {
32         w = i->v;
33         if (reach[w]) continue;
34         else if (dfn[w] == 0)
35         {
36             dfs(w);
37             low[x] = min(low[x], low[w]);
38         }
39         else low[x] = min(low[x], dfn[w]);
40     }
41     if (low[x] == dfn[x])
42     {
43         sccCnt++;
44         do
45         {
46             w = st[top--];
47             idx[w] = sccCnt - 1;
48             reach[w] = true;
49         }while (w != x);
50     }
51 }
52 void toposort(int x)
53 {
54     reach[x] = true;
55     for (edge *i = gscc[x]; i != NULL; i = i->nxt)
56         if (!reach[i->v]) toposort(i->v);

```

```

57     tms[pt++] = x;
58 }
59
60 void build_newgraph(int n)
61 {
62     memset(gsc, 0, sizeof (gsc));
63     for (int i = 0; i < n; i++)
64         for (edge *j = g[i]; j != NULL; j = j->nxt)
65             if (idx[i] != idx[j->v]) addedge(idx[i], idx[j->v], gsc);
66 }
67 void solve(int n)
68 {
69     for (int i = 0; i < n; i++)
70         if (!reach[i]) dfs(i);
71     build_newgraph(n);
72     memset(reach, false, sizeof (reach)); //reuse reach
73     for (int i = 0; i < sccCnt; i++)
74         if (!reach[i]) toposort(i);
75     reverse(tms, tms + pt); //Topological Sort
76 }

```

3.4 2-sat

```

1  #include <iostream>
2  using namespace std;
3  /* 2-sat template node is from 0
4   * i and i^1 is a bool variable(true or false)
5   * conjunctive normal form with 2-sat
6   * x V y == 1 => edge(~x-->y) and edge(~y-->x)
7   * x V y == 0 => (~x V ~x) & (~y V ~y)
8   * x ^ y == (~x V ~y) & (x V y)
9   * x & y == 1 (x V x) & (y V y)
10  * x & y == 0 (~x V ~y) */
11  const int V = 20000, E = 20480 * 4;
12  const int RED = 1, BLUE = 2;
13  struct edge
14  {
15      int v;
16      edge *nxt;
17  } pool[E], *g[V], *pp, *gsc[V];
18  int st[V], top, tms[V], pt;
19  bool reach[V];
20  int dfn[V], low[V], idx[V], sccCnt, depth;
21  int color[V], pre[V];
22  void addedge(int a, int b, edge *g[])
23  {
24      pp->v = b;
25      pp->nxt = g[a];
26      g[a] = pp++;
27  }
28  void initialize()
29  {
30      memset(reach, 0, sizeof (reach));
31      memset(dfn, 0, sizeof (dfn));
32      memset(g, 0, sizeof (g));
33      top = sccCnt = depth = 0, pp = pool;
34  }
35  void dfs(int x)
36  {
37      st[++top] = x;
38      dfn[x] = low[x] = ++depth;
39      int w;
40      for (edge *i = g[x]; i != NULL; i = i->nxt)
41      {

```

```

42     w = i->v;
43     if (reach[w]) continue;
44     else if (dfn[w] == 0)
45     {
46         dfs(w);
47         low[x] = min(low[x], low[w]);
48     }
49     else low[x] = min(low[x], dfn[w]);
50 }
51 if (low[x] == dfn[x])
52 {
53     sccCnt++;
54     do
55     {
56         w = st[top--];
57         idx[w] = sccCnt - 1;
58         reach[w] = true;
59     }while (w != x);
60 }
61 }
62 void toposort(int v)
63 {
64     reach[v] = true;
65     for (edge *i = gscv[v]; i != NULL; i = i->nxt)
66         if (!reach[i->v]) toposort(i->v);
67     tms[pt++] = v;
68 }
69 void build_regraph(int n)//anti-graph
70 {
71     memset(gscv, 0, sizeof (gscv)); //anti-graph scc
72     memset(pre, -1, sizeof (pre)); //the new node to every scc
73     for (int i = 0; i < n; i++)
74     {
75         if (pre[idx[i]] == -1)
76             pre[idx[i]] = i;
77         for (edge * ptr = g[i]; ptr != NULL; ptr = ptr->nxt)
78         {
79             int w = ptr->v;
80             if (idx[i] != idx[w]) addedge(idx[w], idx[i], gscv);
81         }
82     }
83 }
84 void becolor(int v)
85 {
86     color[v] = BLUE;
87     for (edge *i = gscv[v]; i != NULL; i = i->nxt)
88         if (!color[i->v]) becolor(i->v);
89 }
90 void output(int n)//Topological Sort
91 {
92     memset(color, 0, sizeof (color)); //color white
93     for (int i = 0; i < pt; i++)
94     {
95         if (!color[tms[i]]) //color as Topological order
96         {
97             color[tms[i]] = RED;
98             int v = idx[pre[tms[i]] ^ 1];
99             if (color[v] == 0)
100                 becolor(v);
101         }
102     }
103     for (int i = 0; i < n; i += 2)
104     {
105         if (color[idx[i]] == RED)
106             printf("%d\n", i + 1);
107         else //if (color[idx[i ^ 1]] == RED)

```



```

108         printf("%d\n", (i ^ 1) + 1);
109     }
110 }
111 bool solve(int n)//i and ~i can not be in the same scc
112 {
113     for (int i = 0; i < n; i++) if (!reach[i]) dfs(i);
114     for (int i = 0; i < n; i++)
115         if (idx[i] == idx[i ^ 1]) return false;
116     build_regraph(n);
117     pt = 0;
118     memset(reach, 0, sizeof (reach));
119     for (int i = 0; i < sccCnt; i++)
120         if (!reach[i]) toposort(i);
121     reverse(tms, tms + pt);
122     output(n);
123     return true;
124 }
125 int main()
126 {
127     int n, m;
128     while (scanf("%d %d", &n, &m) == 2)
129     {
130         initialize();
131         n *= 2;
132         while (m--)
133         {
134             int a, b;
135             scanf("%d %d", &a, &b);
136             a--, b--;
137             addedge(a, b ^ 1, g);
138             addedge(b, a ^ 1, g);
139         }
140         if (!solve(n)) printf("NIE\n");
141     }
142     return 0;
143 }

```

3.5 Hopcroft-Karp

```

1  const int N = 500, M = 500, INF = 1 << 29;
2  bool g[N][M], chk[M];
3  int Mx[N], My[M], dx[N], dy[M], dis;
4  bool searchP(int n, int m)
5  {
6      queue<int> Q;
7      dis = INF;
8      CC(dx, -1); CC(dy, -1);
9      for (int i = 0; i < n; ++ i)
10         if (Mx[i] == -1)
11         {
12             Q.push(i);
13             dx[i] = 0;
14         }
15     while (!Q.empty())
16     {
17         int u = Q.front();
18         Q.pop();
19         if (dx[u] > dis) break;
20         for (int v = 0; v < m; ++ v)
21             if (g[u][v] && dy[v] == -1)
22             {
23                 dy[v] = dx[u] + 1;
24                 if (My[v] == -1) dis = dy[v];
25                 else

```

```

26         {
27             dx[My[v]] = dy[v] + 1;
28             Q.push(My[v]);
29         }
30     }
31 }
32 return dis != INF;
33 }
34 bool Augment(int u, const int m)
35 {
36     REP(v, 0, m)
37         if (g[u][v] && !chk[v] && dy[v] == dx[u] + 1)
38         {
39             chk[v] = true;
40             if (My[v] != -1 && dy[v] == dis) continue;
41             if (My[v] == -1 || Augment(My[v], m))
42             {
43                 My[v] = u;
44                 Mx[u] = v;
45                 return true;
46             }
47         }
48     return false;
49 }
50 int MaxMatch(int n, int m)
51 {
52     int ans = 0;
53     CC(Mx, -1); CC(My, -1);
54     while (searchP(n, m))
55     {
56         CC(chk, false);
57         REP(i, 0, n)
58             if (Mx[i] == -1 && Augment(i, m)) ++ ans;
59     }
60     return ans;
61 }

```

3.6 hungary

```

1  /*1. simple maximum match
2  2.min path cover of DAG = |V| - max match
3  define: find some edge cover all the nodes
4  build PXP Bipartite graph do the maximum match
5  3.min path cover of Bipartite graph = max match
6  define : find some point cover all the edge(konig)
7  4.chessBoard is a Bipartite graph, then you know
8  5.max independant set(Bipartite graph)=|V| - max match
9  v is all the point of (set A and set B)
10 6.largest cloud(Bipartite graph) = max independant set of Complement*/
11 const int V = 201, E = 10000;
12 vector<int> adj[V];
13 int ym[V], chk[V];
14 bool find_path(int x)
15 {
16     FOREACH(adj[x], i)
17     {
18         if (chk[*i]) continue;
19         chk[*i] = true;
20         if (ym[*i] == -1 || find_path(ym[*i]))
21         {
22             ym[*i] = x;
23             return true;
24         }
25     }

```

```

26     return false;
27 }
28 int slove(int n)
29 {
30     CC(ym, -1);
31     int res = 0;
32     for (int i = 0; i < n; i++)
33     {
34         memset(chk, 0, sizeof (chk));
35         if (find_path(i)) res++;
36     }
37     return res;
38 }

```

3.7 KM

```

1  /* val must be positive
2  * min match use INF - val
3  * must build a matrix[V][V]*/
4  const int V = 100;
5  const int INF = 100000;
6  int val[V][V], lx[V], ly[V], my[V];
7  bool visx[V], visy[V];
8  void initialize(int n)
9  {
10     CC(val, 0), CC(ly, 0), CC(my, -1);
11     fill(lx, lx + n, -INF);
12 }
13 bool find_path(int x, const int n)
14 {
15     visx[x] = true;
16     for(int i = 0; i < n; i++)
17     {
18         if(!visy[i] && lx[x] + ly[i] == val[x][i])
19         {
20             visy[i] = true;
21             if(my[i] == -1 || find_path(my[i], n))
22             {
23                 my[i] = x;
24                 return true;
25             }
26         }
27     }
28     return false;
29 }
30 int solve(int n)
31 {
32     for(int i = 0; i < n; i++)
33         lx[i] = *max_element(val[i], val[i] + n);
34     int dx, sum = 0;
35     for(int i = 0; i < n; i++)
36     {
37         while(true)
38         {
39             CC(visx, 0), CC(visy, 0);
40             if(find_path(i, n)) break;
41             dx = INF;
42             for(int j = 0; j < n; j++)
43             {
44                 if(!visx[j]) continue;
45                 for(int k = 0; k < n; k++)
46                 {
47                     if(visy[k]) continue;
48                     dx = min(dx, lx[j] + ly[k] - val[j][k]);

```

```

49         }
50     }
51     for(int j = 0; j < n; j++)
52     {
53         if(visx[j]) lx[j] -= dx;
54         if(visy[j]) ly[j] += dx;
55     }
56 }
57 }
58 for(int i = 0; i < n; i++)
59     sum += INF - val[my[i]][i];
60 return sum;
61 }

```

3.8 stableMarriage

```

1  /* boy[i][j] gg[i] to mm[j]
2  * girl[i][j] mm[i] to gg[j]*/
3  const int N = 26;
4  const int M = 128;
5  int boy[N][N], girl[N][N];
6  int my[N], mx[N], now[N];
7  void Gale_Shapley(int n)
8  {
9      queue<int> q;
10     for(int i = 0; i < n; i++) q.push(i);
11     while(!q.empty())
12     {
13         int i = q.front(); q.pop();
14         int j = now[i]++, mm = boy[i][j];
15         if(my[mm] == -1 || girl[mm][my[mm]] > girl[mm][i])
16         {
17             if(my[mm] != -1) q.push(my[mm]);
18             my[mm] = i, mx[i] = mm;
19         }
20         else q.push(i);
21     }
22 }
23
24 char nameB[N], nameG[N];
25 void output(int n)
26 {
27     for(int i = 0; i < n; i++)
28         printf("%c %c\n", nameB[i], nameG[mx[i]]);
29 }
30
31 int hashB[M], hashG[M];
32 void initialize()
33 {
34     memset(hashB, 0, sizeof(hashB)), memset(hashG, 0, sizeof(hashG));
35     memset(my, -1, sizeof(my)), memset(now, 0, sizeof(now));
36 }

```

3.9 maximal Clique

```

1  /* 求无向图极大团的个数
2  * 极大团就一个被不被其他的完全子图包含的完全子图
3  * 最大团一定是一个极大团，但是极大团不一定是最大团
4  */
5  class Bron_Kerbosch
6  {
7  private:

```

```

8      const static int N = 130;
9      int n, maps[N][N], cnt;
10     void countClique(int *p, int ps, int *x, int xs)
11     {
12         if(ps == 0)
13         {
14             if(xs == 0)
15                 cnt++;
16             return ;
17         }
18         for(int i = 0; i < xs; i++)
19         {
20             int j, v = x[i];
21             for(j = 0; j < ps && maps[p[j]][v]; j++);
22             if(j == ps)
23                 return;
24         }
25         int tmpp[N], tmpps = 0, tmpx[N], tmpxs = 0;
26         for(int i = 0; i < ps; i++)
27         {
28             int v = p[i];
29             tmpps = tmpxs = 0;
30             for(int j = i + 1; j < ps; j++)
31             {
32                 int u = p[j];
33                 if(maps[v][u])
34                     tmpp[tmpps++] = u;
35             }
36             for(int j = 0; j < xs; j++)
37             {
38                 int u = x[j];
39                 if(maps[v][u])
40                     tmpx[tmpxs++] = u;
41             }
42             countClique(tmpp, tmpps, tmpx, tmpxs);
43             if(cnt > 1000)
44                 return;
45             x[xs++] = v;
46         }
47     }
48     public:
49     void initialize(int n, int m)
50     {
51         memset(maps, 0, sizeof(maps));
52         this->n = n;
53         for(int i = 0; i < m; i++)
54         {
55             int a, b;
56             scanf("%d %d", &a, &b);
57             a--, b--;
58             maps[a][b] = true;
59             maps[b][a] = true;
60         }
61     }
62     int countClique()
63     {
64         cnt = 0;
65         int p[N], x[N];
66         for(int i = 0; i < n; i++)
67             p[i] = i;
68         countClique(p, n, x, 0);
69         return cnt;
70     }
71 }one ;

```

3.10 MaxClique

```
1  const int N = 50;
2  int maps[N][N], found, mc, n;
3  int c[N], answer[N], record[N];
4  void dfs(int GraphSize, int *s, int CliqueSize)
5  {
6      if(GraphSize == 0)
7      {
8          if(CliqueSize > mc)
9          {
10             mc = CliqueSize;
11             found = true;
12             copy(record, record + mc, answer);
13         }
14         return ;
15     }
16     for(int i = 0; i < GraphSize; i++)
17     {
18         if(CliqueSize + GraphSize <= mc || c[s[i]] + CliqueSize <= mc)
19             return;
20         int tmps[N], tmpSize = 0;
21         record[CliqueSize] = s[i];
22         for(int j = i + 1; j < GraphSize; j++)
23             if(maps[s[i]][s[j]])
24                 tmps[tmpSize++] = s[j];
25         dfs(tmpSize, tmps, CliqueSize + 1);
26         if(found)
27             return ;
28     }
29 }
30 void initialize()
31 {
32     memset(maps, false, sizeof(maps));
33     mc = 0;
34 }
35 int findMaxClique(int n)
36 {
37     for(int i = n - 1; i >= 0; i--)
38     {
39         found = false;
40         int tail = 0, s[N];
41         for(int j = i + 1; j < n; j++)
42             if(maps[i][j])
43                 s[tail++] = j;
44         record[0] = i;
45         dfs(tail, s, 1);
46         c[i] = mc;
47     }
48     return mc;
49 }
```

3.11 minimum Cut

```
1  const int V = 501, INF = 100000000, S = 1;
2  int maps[V][V], dist[V], pre;
3  bool vst[V], del[V];
4  void initialize()// start with 1
5  {
6      memset(del, false, sizeof (del));
7      memset(maps, 0, sizeof (maps));
8  }
9  int maximum_adjacency_search(int t, int n)
10 {
```

```

11     for (int i = 1; i <= n; i++)
12         if (!del[i]) dist[i] = maps[S][i];
13     memset(vst, false, sizeof (vst));
14     vst[S] = true;
15     int k = S;
16     for (int j = 1; j <= n - t; j++)
17     {
18         int tmp = -INF;
19         pre = k;
20         for (int i = 1; i <= n; i++)
21             if (!vst[i] && !del[i] && tmp < dist[i])
22             {
23                 tmp = dist[i];
24                 k = i;
25             }
26         vst[k] = true;
27         for (int i = 1; i <= n; i++)
28             if (!vst[i] && !del[i]) dist[i] += maps[k][i];
29     }
30     return k;
31 }
32 int Stoer_Wgner(int n)
33 {
34     int mcut = INF;
35     for (int i = 1; i < n; i++)
36     {
37         int idx = maximum_adjacency_search(i, n);
38         mcut = min(mcut, dist[idx]);
39         del[idx] = true;
40         for (int i = 1; i <= n; i++)
41         {
42             if (!del[i] && i != pre)
43             {
44                 maps[pre][i] += maps[idx][i];
45                 maps[i][pre] = maps[pre][i];
46             }
47         }
48     }
49     return mcut;
50 }

```

3.12 LCA

```

1  铭稿
2  onst int N = 100000;
3  int father[N], chk[N], dgr[N];
4  vector<vector<int>> adj, query;
5  int set_find(int i)
6  {
7      return father[i] = i == father[i] ? i : set_find(father[i]);
8  }
9  void initialize(int n)
10 {
11     adj.assign(n, vector<int>());
12     query.assign(n, vector<int>());
13     CC(dgr, 0); CC(chk, 0);
14 }
15
16 void LCA(int u)
17 {
18     father[u] = u;
19     FOREACH(adj[u], i)
20     {
21         LCA(*i), father[*i] = u;

```

```

22     }
23     chk[u] = 1;
24     FOREACH(query[u], i) if(chk[*i])
25         printf("%d\n", set_find(*i));
26 }

```

3.13 bellman-ford

```

1  #include <iostream>
2  #include <cstring>
3  #include <cstdio>
4  #include <vector>
5  using namespace std;
6  const int SIZE = 10110;
7  const int INF = 1000000000;
8  vector<pair<pair<int, int>,int> >edge;
9  int n, a0, a1, b0, b1, l0, l1;
10 int d[SIZE];
11
12 void Bellman_ford(int n,int p)
13 {
14     fill(d,d + n, INF);
15     d[n - 1] = 0;
16     for(int i = 1;i < n;i++)
17     {
18         bool unfind = true;
19         FOREACH(edge, i)
20         {
21             int v = i->first.first, u = i->first.second, val = i->second;
22             if(d[u] > d[v] + val)
23             {
24                 unfind = false;
25                 d[u] = d[v] + val;
26             }
27         }
28         if(unfind)
29             break;
30     }
31     FOREACH(edge, i)
32     {
33         int v = i->first.first, u = i->first.second, val = i->second;
34         if(d[u] > d[v] + val)
35         {
36             puts("-1");
37             return;
38         }
39     }
40     for(int i = 1;i <= p;i++)
41         printf("%d", d[i] - d[i - 1]);
42     puts("");
43 }
44 int main()
45 {
46     while(scanf("%d%d%d%d%d%d", &n, &a0, &b0, &l0, &a1, &b1, &l1) == 7)
47     {
48         edge.clear();
49         for(int i = 1;i + l0 - 1 <= n;i++)
50         {
51             edge.push_back(make_pair(make_pair(i-1,i+l0-1),l0-a0));
52             edge.push_back(make_pair(make_pair(i+l0-1,i-1),b0 - l0));
53         }
54         for(int i = 1;i + l1 - 1 <= n;i++)
55         {
56             edge.push_back(make_pair(make_pair(i-1,i+l1-1),b1));

```



```

57         edge.push_back(make_pair(make_pair(i+l1-1,i-1),-a1));
58     }
59     for(int i = 1;i <= n;i++)
60     {
61         edge.push_back(make_pair(make_pair(i - 1, i), 1));
62         edge.push_back(make_pair(make_pair(i, i - 1), 0));
63     }
64     for(int i = 0;i <= n;i++)
65         edge.push_back(make_pair(make_pair(n + 1, i), 0));
66     Bellman_ford(n + 2, n);
67 }
68 return 0;
69 }

```

3.14 Euler-fleury

```

1  锺縞
2  const int SIZE = 2 * 2000;
3  const int N = 50;
4  /* Euler degree & connection
5   * fordown 研究paths the smallest lexicographic path
6   * hoj 1045 John's trip*/
7  struct edge
8  {
9      int v, id;
10     bool operator<(const edge a) const
11     {
12         return id < a.id;
13     }
14 } edges[SIZE];
15 vector<edge> adj[N];
16 int path[SIZE];
17 int E, V, S, deg[N], stp;
18 bool vst[SIZE];
19
20 void dfs(int now)
21 {
22     edge tmp;
23     for (size_t i = 0; i < adj[now].size(); i++)
24     {
25         tmp = adj[now][i];
26         if (!vst[tmp.id] && !vst[tmp.v])
27         {
28             vst[tmp.id] = vst[tmp.v] = 1;
29             dfs(tmp.v);
30             path[stp++] = tmp.id;
31         }
32     }
33 }
34
35 void solve()
36 {
37     /*if (!check())
38         printf("Round trip does not exist.\n");
39     else*/
40     {
41         for (int i = 0; i < V; i++)
42             sort(adj[i].begin(), adj[i].end());
43         dfs(S);
44         printf("%d", path[stp - 1]);
45         for (int i = stp - 2; i >= 0; i--)
46             printf(" %d", path[i]);
47         putchar('\n');
48     }

```

```

49 }
50
51 void initialize(int u, int v)
52 {
53     stp = V = E = S = 0;
54     for (int i = 0; i < N; i++) adj[i].clear();
55     memset(vst, false, sizeof (vst));
56     memset(deg, 0, sizeof (deg));
57     S = min(v, u);
58 }
59
60 void add_edge(int u, int v, int id, int E)
61 {
62     deg[u]++, deg[v]++;
63     edges[E].v = v, edges[E].id = id;
64     adj[u].push_back(edges[E]);
65     edges[E].v = u, edges[E].id = id;
66     adj[v].push_back(edges[E]);
67 }

```

3.15 k-shortest-path(with cycle)

```

1  /* 估价函数f(x)=g(x)+h(x);h(x)=h*(x) 所以符合A条件*, 短路含环k
2  * 中为statusfA的估价函数。*f(x2)=f(x1)-h(x1)+h(x2)+c(x1)(x2);*/
3  const int V = 1000, E = 100000, INF = 1000000000;
4  struct status
5  {
6      int v, f;
7      status() {}
8      status(int _v, int _f)
9      {v = _v, f = _f;}
10     bool operator <(const status a) const
11     {return f > a.f;}
12 };
13
14 struct edge
15 {
16     int v, dist;
17     edge *nxt;
18 }*pp, *g[V], *rg[V], pool[E * 2];
19 int d[V], c[V];
20 bool chk[V];
21 void initialize()
22 {
23     pp = pool;
24     memset(g, 0, sizeof(g));
25     memset(rg, 0, sizeof(rg));
26 }
27
28 void addedge(int u, int v, int dist, edge *g[])
29 {
30     pp->v = v;
31     pp->dist = dist;
32     pp->nxt = g[u];
33     g[u] = pp++;
34 }
35
36 void dijkstra(int n, int t)
37 {
38     fill(d, d + n, INF);
39     memset(chk, false, sizeof(chk));
40     priority_queue<status> pq;
41     pq.push(status(t, 0));
42     while(!pq.empty())

```

```

43     {
44         status now = pq.top();
45         pq.pop();
46         if(chk[now.v]) continue;
47         chk[now.v] = true;
48         d[now.v] = now.f;
49         for(edge *i = rg[now.v]; i != NULL; i = i->nxt)
50             pq.push(status(i->v, now.f + i->dist));
51     }
52 }
53
54 int Astar(int s, int t, int k)
55 {
56     if(d[s] == INF) return -1;
57     memset(c, 0, sizeof(c));
58     priority_queue<status> pq;
59     pq.push(status(s, d[s]));
60     while(!pq.empty())
61     {
62         status now = pq.top();
63         pq.pop();
64         c[now.v]++;
65         if(c[t] == k) return now.f;
66         if(c[now.v] > k) continue;
67         for(edge *i = g[now.v]; i != NULL; i = i->nxt)
68             pq.push(status(i->v, now.f - d[now.v] + d[i->v] + i->dist));
69     }
70     return -1;
71 }
72
73 int main()
74 {
75     int n, m, s, t, k;
76     int u, v, dist;
77     while(scanf("%d %d", &n, &m) == 2)
78     {
79         initialize();
80         for(int i = 0; i < m; i++)
81         {
82             scanf("%d %d %d", &u, &v, &dist);
83             u--, v--;
84             addedge(u, v, dist, g);
85             addedge(v, u, dist, rg);
86         }
87         scanf("%d %d %d", &s, &t, &k);
88         s--, t--;
89         if(s == t) k++;
90         dijkstra(n, t);
91         printf("%d\n", Astar(s, t, k));
92     }
93     return 0;
94 }

```

3.16 secondSP

```

1  /*由最短路推出答案,算法变形dijkstra
2  * 本题求了最短路以及比最短路大的路的条数。1
3  * poj 3463 Sightseeing
4  * 本模板允许向回走,再次走到原来经过的点
5  * 如果要求不课重复走的次短路,记录最短路前驱
6  * 然后枚举删掉最短路上的边,再求最短路取最小值便是次短路*/
7  const int SIZE = 1010;
8  const int INF = 1100000000;
9  class Edge

```

```

10 {
11 public:
12     int v, cost, pt;
13
14     bool operator<(const Edge a) const
15     {
16         return cost > a.cost;
17     }
18 } tmp, nxt, beg;
19 vector<Edge> adj[SIZE];
20 int dist[SIZE][2], cnt[SIZE][2];
21 int n, m;
22 bool check[SIZE][2];
23
24 int dijkstra(int start, int end)
25 {
26     memset(check, false, sizeof (check));
27     memset(cnt, 0, sizeof (cnt));
28     for (int i = 1; i <= n; i++)
29         dist[i][0] = dist[i][1] = INF;
30     priority_queue<Edge> pq;
31     beg.v = start, beg.pt = 0, beg.cost = 0;
32     pq.push(beg);
33     cnt[start][0] = 1;
34     dist[start][0] = 0;
35     while (!pq.empty())
36     {
37         beg = pq.top();
38         pq.pop();
39         if (check[beg.v][beg.pt])
40             continue;
41         check[beg.v][beg.pt] = true;
42         int s = adj[beg.v].size(), w, cst;
43         for (int i = 0; i < s; i++)
44         {
45             tmp = adj[beg.v][i];
46             w = tmp.v, cst = tmp.cost;
47             if (dist[w][0] > dist[beg.v][beg.pt] + cst && !check[w][0])
48             {
49                 dist[w][1] = dist[w][0];
50                 cnt[w][1] = cnt[w][0];
51                 dist[w][0] = dist[beg.v][beg.pt] + cst;
52                 cnt[w][0] = cnt[beg.v][beg.pt];
53                 nxt.v = w, nxt.pt = 0, nxt.cost = dist[w][0];
54                 pq.push(nxt);
55                 nxt.v = w, nxt.pt = 1, nxt.cost = dist[w][1];
56                 pq.push(nxt);
57             }
58             else if (dist[w][0] == dist[beg.v][beg.pt] + cst && !check[w][0])
59             {
60                 cnt[w][0] += cnt[beg.v][beg.pt];
61                 nxt.v = w, nxt.pt = 0, nxt.cost = dist[w][0];
62                 pq.push(nxt);
63             }
64             else if (dist[w][1] > dist[beg.v][beg.pt] + cst && !check[w][1])
65             {
66                 dist[w][1] = dist[beg.v][beg.pt] + cst;
67                 cnt[w][1] = cnt[beg.v][beg.pt];
68                 nxt.pt = 1, nxt.v = w, nxt.cost = dist[w][1];
69                 pq.push(nxt);
70             }
71             else if (dist[w][1] == dist[beg.v][beg.pt] + cst && !check[w][1])
72             {
73                 cnt[w][1] += cnt[beg.v][beg.pt];
74                 nxt.pt = 1, nxt.v = w, nxt.cost = dist[w][1];
75                 pq.push(nxt);

```

```

76         }
77     }
78 }
79 int num = cnt[end][0];
80 if (dist[end][0] + 1 == dist[end][1])
81     num += cnt[end][1];
82 return num;
83 }
84
85 int main()
86 {
87     int T;
88     scanf("%d", &T);
89     while (T--)
90     {
91         scanf("%d %d", &n, &m);
92         for (int i = 1; i <= n; ++i)
93             adj[i].clear();
94         for (int i = 0; i < m; ++i)
95         {
96             int a, b, c;
97             scanf("%d %d %d", &a, &b, &c);
98             beg.v = b, beg.cost = c;
99             adj[a].push_back(beg);
100         }
101         int start, end;
102         scanf("%d %d", &start, &end);
103         printf("%d\n", dijkstra(start, end));
104     }
105     return 0;
106 }

```

3.17 SPFA

```

1  /* 模板，队列实现SPFA
2  * 如果要求负环，设置人工顶点
3  * 与已知点连边权值为0
4  * 人工顶点先入队列
5  * 记录每个点入队列的次数，如果大于则有负环n
6  */
7  const int V = 1010, E = 300001, INF = 100000000;
8  struct edges
9  {
10     int v, val;
11     edges *next;
12 } pool[E], *g[V], *pp;
13 int cnt[V], dist[V];
14 bool vst[V];
15 void initialize()
16 {
17     memset(g, 0, sizeof (g));
18     pp = pool;
19 }
20 void addedge(int a, int b, int v)
21 {
22     pp->v = b;
23     pp->val = v;
24     pp->next = g[a];
25     g[a] = pp++;
26 }
27 void SPFA(int n)
28 {
29     memset(cnt, 0, sizeof (cnt));
30     memset(vst, false, sizeof (vst));

```

```

31     fill(dist, dist + n, INF);
32     queue<int> q;
33     q.push(n - 1);
34     dist[n - 1] = 0;
35     vst[n - 1] = true;
36     while (!q.empty())
37     {
38         int beg = q.front();
39         q.pop();
40         vst[beg] = false;
41         for (edges * i = g[beg]; i != NULL; i = i->next)
42         {
43             int tmp = i->v, val = i->val;
44             if (dist[tmp] > dist[beg] + val)
45             {
46                 dist[tmp] = dist[beg] + val;
47                 if (!vst[tmp])
48                 {
49                     vst[tmp] = true;
50                     cnt[tmp]++;
51                     q.push(tmp);
52                     if (cnt[tmp] > n)
53                     {
54                         puts("-1");
55                         return;
56                     }
57                 }
58             }
59         }
60     }
61     for(int i = 1; i < n - 1; i++)
62         printf("%d", dist[i] - dist[i - 1]);
63     puts("");
64 }
65 int main()
66 {
67     int n, a0, b0, l0, a1, b1, l1;
68     while(scanf("%d %d %d %d %d %d", &n, &a0, &b0, &l0, &a1, &b1, &l1) == 7)
69     {
70         initialize();
71         for(int i = 1; i + l0 - 1 <= n; i++)
72         {
73             addedge(i - 1, i + l0 - 1, l0 - a0);
74             addedge(i + l0 - 1, i - 1, b0 - l0);
75         }
76         for(int i = 1; i + l1 - 1 <= n; i++)
77         {
78             addedge(i - 1, i + l1 - 1, b1);
79             addedge(i + l1 - 1, i - 1, -a1);
80         }
81         for(int i = 1; i <= n; i++)
82         {
83             addedge(i - 1, i, 1);
84             addedge(i, i - 1, 0);
85         }
86         for(int i = 0; i <= n; i++)
87             addedge(n + 1, i, 0);
88         SPFA(n + 2);
89     }
90     return 0;
91 }

```

3.18 Best Radio Spanning Tree

```

1  const int SIZE = 1001;
2  const double EXP = 1e-5;
3  const double INF = 1e10;
4  double mat[SIZE][SIZE], cost[SIZE][SIZE], val[SIZE][SIZE];
5  int n;
6  double x[SIZE], y[SIZE], height[SIZE];
7  double prim(double low)
8  {
9      bool check[SIZE];
10     memset(check, false, sizeof (check));
11     check[0] = true;
12     double totdis = 0, totcost = 0, dist[SIZE], minn = INF;
13     dist[0] = 0;
14     for (int i = 1; i < n; i++)
15         dist[i] = INF;
16     int idx = 0, pree[SIZE], tmp;
17     memset(pree, 0, sizeof (pree));
18     for (int i = 0; i < n; i++)
19         for (int j = 0; j < n; j++)
20             val[i][j] = cost[i][j] - low * mat[i][j];
21     for (int i = 1; i < n; i++)
22     {
23         minn = INF;
24         for (int j = 1; j < n; j++)
25         {
26             if (!check[j])
27             {
28                 if (dist[j] > val[idx][j])
29                 {
30                     dist[j] = val[idx][j];
31                     pree[j] = idx;
32                 }
33                 if (dist[j] < minn)
34                 {
35                     minn = dist[j];
36                     tmp = j;
37                 }
38             }
39         }
40         totdis += mat[pree[tmp]][tmp];
41         totcost += cost[pree[tmp]][tmp];
42         check[tmp] = true;
43         idx = tmp;
44     }
45     return totcost / totdis;
46 }
47 double distance(int i, int j)
48 {
49     return sqrt(pow(x[i] - x[j], 2) + pow(y[i] - y[j], 2));
50 }
51 int main()
52 {
53     while (scanf("%d", &n) == 1 && n)
54     {
55         for (int i = 0; i < n; i++)
56             scanf("%lf %lf %lf", &x[i], &y[i], &height[i]);
57         for (int i = 0; i < n; i++)
58             for (int j = 0; j < n; j++)
59             {
60                 mat[i][j] = distance(i, j);
61                 cost[i][j] = fabs(height[i] - height[j]);
62             }
63         double low = 0, tmp;
64         while (true)
65         {
66             tmp = prim(low);

```

```

67         if (fabs(low - tmp) < EXP)
68             break;
69         low = tmp;
70     }
71     printf("%.3lf\n", tmp);
72 }
73 return 0;
74 }

```

3.19 Count Spanning Tree

```

1  const int SIZE = 12;
2  double guass(int n, double mat[][SIZE])
3  {
4      for (int i = 1; i < n; i++)
5      {
6          for (int j = 0; j < i; j++)
7          {
8              if (mat[i][j] == 0)
9                  continue;
10             double kk = mat[i][j] / mat[j][j];
11             for (int k = 0; k < n; k++)
12                 mat[i][k] -= kk * mat[j][k];
13         }
14     }
15     double res = 1.00;
16     for(int i = 0; i < n; i++)
17         res *= mat[i][i];
18     return fabs(res) + 0.005;
19 }
20 double maps[SIZE][SIZE];
21 int main()
22 {
23     int n, t;
24     scanf("%d", &t);
25     while (t--)
26     {
27         scanf("%d", &n);
28         for (int i = 0; i < n; i++)
29         {
30             int cnt = 0;
31             for (int j = 0; j < n; j++)
32             {
33                 scanf("%lf", &maps[i][j]);
34                 if(i == j)
35                     maps[i][j] = 0;
36                 if (maps[i][j] == 1)
37                     cnt++;
38                 maps[i][j] = -maps[i][j];
39             }
40             maps[i][i] = cnt;
41         }
42         printf("%.0lf\n", guass(n - 1, maps));
43     }
44     return 0;
45 }

```

3.20 Degree Limited Spanning Tree

```

1  /* 度限制最小生成树为根0 限制度最大k */
2  const int N = 25;
3  const int LEN = 15;

```



```

4  const int INF = 1<<29;
5  int dis[N][N]= {}, f[N]= {}, father[N]= {}, n;
6  bool visit[N]= {};
7  bool used[N][N]= {};
8  void Dfs(int last, int v)//node 0 is root
9  {
10     visit[v] = 1;
11     if (!father[v]) f[v] = -INF;
12     else f[v] = max(dis[last][v], f[father[v]]);
13     for (int i = 0; i < n; ++i)
14         if (!visit[i] && used[v][i])
15             father[i] = v, Dfs(v, i);
16 }
17 int DegreeLimitMST(int k)
18 {
19     int ret = 0, path[N], group[N]= {}, g = 0, pre[N], degree = 0;
20     memset(used, 0, sizeof(used));
21     for (int i = 1; i < n; ++i)//除了点的最小生成森林0
22         if (!group[i])
23         {
24             group[i] = ++g;
25             for (int j = 0; j < n; ++j)
26                 path[j] = dis[i][j], pre[j] = i;
27             while (1)
28             {
29                 int tmp = INF, mark = -1;
30                 for (int j = 1; j < n; ++j)
31                     if (!group[j] && path[j] < tmp)
32                         tmp = path[j], mark = j;
33                 if (mark == -1) break;
34                 used[pre[mark]][mark] = 1, used[mark][pre[mark]] = 1;
35                 ret += tmp;
36                 group[mark] = g;
37                 for (int j = 1; j < n; ++j)
38                     if (!group[j] && path[j] > dis[mark][j])
39                         path[j] = dis[mark][j], pre[j] = mark;
40             }
41         }
42     for (int i = 1; i <= g; ++i)//和点相连0
43     {
44         int tmp = INF, mark = -1;
45         for (int j = 1; j < n; ++j)
46             if (group[j] == i && tmp > dis[0][j])
47                 tmp = dis[0][j], mark = j;
48         used[0][mark] = used[mark][0] = 1;
49         ret += tmp;
50         ++degree;
51     }
52     while (degree < k)//保证有解不可能森林大于,个,通过增大度减少树的边权k
53     {
54         memset(visit, 0, sizeof(visit));
55         Dfs(0, 0);
56         int tmp = INF, mark = -1, t;
57         for (int i = 1; i < n; ++i)
58             if (!used[0][i] && dis[0][i] != INF)
59             {
60                 t = ret+dis[0][i]-f[i];
61                 if (tmp > t) tmp = t, mark = i;
62             }
63         if (ret <= tmp) break;
64         ret = tmp;
65         used[0][mark] = used[mark][0] = 1;
66         tmp = f[mark];
67         while (dis[father[mark]][mark] != tmp) mark = father[mark];
68         used[mark][father[mark]] = used[father[mark]][mark] = 0;
69         ++degree;

```

```

70     }
71     return ret;
72 }

```

3.21 second MST

```

1  /*算法，并查集
2  kruskal统计了每个点在中出现次数
3  mst求次小生成树，是最小生成树的邻集
4
5  n求每个点之间的最小边，然后替代之^22
6  */
7  #include <iostream>
8  #include <cstdio>
9  #include <cstring>
10 #include <algorithm>
11 #include <vector>
12 using namespace std;
13 const int SIZE = 101;
14 const int INF = 0x7fffffff;
15
16 struct Edge
17 {
18     int v, w, val;
19
20     bool operator<(const Edge a) const
21     {
22         return val < a.val;
23     }
24 } edge[SIZE * SIZE];
25 bool check[SIZE], used[SIZE * SIZE];
26 int father[SIZE], rank[SIZE];
27 int max_val[SIZE][SIZE], mst;
28 vector<Edge> adj[SIZE];
29 int set_find(int x)
30 {
31     return father[x] = father[x] == x ? father[x] : set_find(father[x]);
32 }
33
34 bool set_link(int x, int y)
35 {
36     if (x == y)
37         return false;
38     if (rank[x] < rank[y])
39         father[x] = y;
40     else
41     {
42         father[y] = x;
43         rank[x] += rank[x] == rank[y];
44     }
45     return true;
46 }
47
48 bool set_union(int x, int y)
49 {
50     return set_link(set_find(x), set_find(y));
51 }
52 int t, n, m;
53
54 void initialize()
55 {
56     scanf("%d %d", &n, &m);
57     for (int i = 1; i <= n; i++)
58     {

```

```

59     adj[i].clear();
60     father[i] = i;
61     rank[i] = 0;
62 }
63 mst = 0;
64 int v, w, val;
65 for (int i = 0; i < m; i++)
66 {
67     scanf("%d %d %d", &v, &w, &val);
68     edge[i].v = v;
69     edge[i].w = w;
70     edge[i].val = val;
71 }
72 memset(check, false, sizeof (check));
73 memset(used, false, sizeof(used));
74 }
75 void kruskal()
76 {
77     for (int i = 0; i < m; i++)
78     {
79         if(set_union(edge[i].v, edge[i].w))
80         {
81             Edge temp = edge[i];
82             adj[edge[i].v].push_back(temp);
83             temp.w = temp.v;
84             adj[edge[i].w].push_back(temp);
85             mst += edge[i].val;
86             used[i] = true;
87         }
88     }
89 }
90 void dfs(int now, int pre, int maxn, int val)
91 {
92     check[now] = true;
93     max_val[pre][now] = max(maxn, val);
94     int s = adj[now].size(), w;
95     for(int i = 0; i < s; i++)
96     {
97         w = adj[now][i].w;
98         if(!check[w])
99         {
100             dfs(w, pre, max_val[pre][now], adj[now][i].val);
101         }
102     }
103 }
104 bool sst()
105 {
106     for(int i = 0; i < m; i++)
107     {
108         if(!used[i])
109         {
110             if(edge[i].val == max_val[edge[i].v][edge[i].w])
111                 return false;
112         }
113     }
114     return true;
115 }
116 int main()
117 {
118     scanf("%d", &t);
119     while (t--)
120     {
121         initialize();
122         sort(edge, edge + m);
123         kruskal();
124         for(int i = 1; i <= n; i++)

```

```

125     {
126         int s = adj[i].size(),w;
127         memset(check,false,sizeof(check));
128         check[i] = true;
129         for(int j = 0; j < s;j++)
130         {
131             w = adj[i][j].w;
132             dfs(w,i,0,adj[i][j].val);
133         }
134     }
135     if(sst())
136         printf("%d\n",mst);
137     else
138         printf("Not Unique!\n");
139 }
140 return 0;
141 }

```

3.22 minimum direct tree(matrix)

```

1  锳綯
2  const int SIZE = 1001;
3  const int INF = 1000000000;
4  double g[SIZE][SIZE], x[SIZE], y[SIZE];
5  bool visit[SIZE], circle[SIZE];
6  int pre[SIZE];
7  double dist(int i, int j)
8  {
9      return sqrt(pow((x[i]-x[j]),2)+pow((y[i]-y[j]),2));
10 }
11 void dfs(int t, int n)
12 {
13     if (visit[t]) return;
14     visit[t] = 1;
15     REP(i, 0, n) if (g[t][i] < INF) dfs(i, n);
16 }
17 bool connect(int root, int n)
18 { //judge contection
19     CC(visit, 0);dfs(root, n);
20     return accumulate(visit, visit + n, 0) == n;
21 }
22 void findMinimumEdge(const int root, int n)
23 { //find min edge of every edge
24     REP(i, 0, n)
25     {
26         if (circle[i] || i == root) continue;
27         pre[i] = i;
28         double tmp = INF + 1;
29         REP(j, 0, n)
30         {
31             if (circle[j]) continue;
32             if (g[j][i] < tmp && i != j)
33             {
34                 tmp = g[j][i];
35                 pre[i] = j;
36             }
37         }
38     }
39 }
40 int findCrile(const int root, int n)
41 {
42     REP(i, 0, n)
43     {
44         if(circle[i]) continue;

```

```

45     int now = i;
46     CC(visit, 0);
47     while(!visit[now] && now != root)
48     {
49         visit[now] = true;
50         now = pre[now];
51     }
52     if(now != root) return now;
53 }
54 return -1;
55 }
56 void update(int now, int n)
57 {
58     REP(j, 0, n)
59     {
60         if (circle[j]) continue;
61         if (g[j][now] < INF)
62             g[j][now] -= g[pre[now]][now]; //update inEdge
63     }
64     for (int j = pre[now]; j != now; j = pre[j])
65     {
66         REP(k, 0, n)
67         {
68             if (circle[k]) continue;
69             if (g[j][k] < INF) //update outEdge
70                 g[now][k] = min(g[now][k], g[j][k]);
71             if (g[k][j] < INF) //update inEdge
72                 g[k][now] = min(g[k][now], g[k][j] - g[pre[j]][j]);
73         }
74     }
75 }
76 double solve(const int root, int n)
77 {
78     double ans = 0;
79     int now;
80     memset(circle, 0, sizeof (circle));
81     do
82     {
83         findMinimumEdge(root, n);
84         if((now = findCrile(root, n)) != -1)
85         {
86             ans += g[pre[now]][now];
87             for (int j = pre[now]; j != now; j = pre[j])
88             {
89                 ans += g[pre[j]][j];
90                 circle[j] = 1;
91             }
92             update(now, n);
93         }
94         else
95         {
96             REP(j, 0, n)
97             {
98                 if (circle[j] || j == root) continue;
99                 ans += g[pre[j]][j];
100             }
101         }
102     }while(now != -1);
103     return ans;
104 }
105 int main()
106 {
107     int n, m, a, b;
108     while (scanf("%d%d", &n, &m) != EOF)
109     {
110         REP(i, 0, n) scanf("%lf%lf", &x[i], &y[i]);

```

```

111     REP(i, 0, n) REP(j, 0, n) g[i][j] = INF;
112     REP(i, 0, m)
113     {
114         scanf("%d%d", &a, &b);
115         a--, b--;
116         g[a][b] = dist(a, b);
117     }
118     if (!connect(0, n)) printf("poor snoopy\n");
119     else printf("%.2lf\n", solve(0, n));
120 }
121 return 0;
122 }

```

3.23 minimum direct tree(pool)

```

1  const int N = 1010;
2  const int E = N * N;
3  const LL INF = 100000000000LL;
4  template<typename T>
5  struct Edge
6  {
7      int u, v;
8      T c;
9  };
10 Edge<LL> edge[E];
11 int label[N], pre[N], visit[N];
12 template<typename T>
13 T treeGraph(int n, int m, int root, Edge<T>* edge)
14 {
15     int cnt = 0;
16     T inEdge[N], ans = 0;
17     while(true)
18     {
19         fill(inEdge, inEdge + n, INF);
20         REP(i, 0, m)
21         {
22             int u = edge[i].u;
23             int v = edge[i].v;
24             if(v != u && edge[i].c < inEdge[v])
25             {
26                 pre[v] = u;
27                 inEdge[v] = edge[i].c;
28             }
29         }
30         REP(i, 0, n)
31         {
32             if(i == root) continue;
33             if(inEdge[i] == INF) return -1;
34         }
35         int now = 0;
36         CC(label, -1);
37         CC(visit, -1);
38         inEdge[root] = 0;
39         REP(i, 0, n)
40         {
41             ans += inEdge[i];
42             int v = i;
43             while(visit[v] != i && label[v] == -1 && v != root)
44             {
45                 visit[v] = i;
46                 v = pre[v];
47             }
48             if(v != root && label[v] == -1)
49             {

```

```

50         for(int u = pre[v]; u != v; u = pre[u])
51             label[u] = now;
52         label[v] = now++;
53     }
54 }
55 if(now == 0) break;
56 REP(i, 0, n) if(label[i] == -1) label[i] = now++;
57 REP(i, 0, m)
58 {
59     int v = edge[i].v;
60     edge[i].v = label[edge[i].v];
61     edge[i].u = label[edge[i].u];
62     if(edge[i].v != edge[i].u) edge[i].c -= inEdge[v];
63 }
64 root = label[root];
65 n = now;
66 }
67 return ans;
68 }

```

3.24 maxflow

```

1 #define OP(i) (((i) - (pool))^1)
2 class sap
3 {
4 private:
5     const static int V = 20010, E = 1000000, INF = 100000000;
6     int dis[V], numdis[V], pre[V], maxflow;;
7     bool reachS[V], reachT[V];
8     struct edge
9     {
10         int v, cap;
11         edge *nxt; //保存当前弧, 存可行流中的边
12     } pool[E], *g[V], *pp, *e[V], *pree[V];
13     void bfs(int v, int n) //从汇点开始按照反向边流量走
14     {
15         int que[V], tail = 0;
16         bool vst[V] = {0};
17         memset(numdis, 0, sizeof(numdis));
18         fill(dis, dis + n, n);
19         dis[v] = 0, vst[v] = 1, que[0] = v;
20         for (int j = 0; j <= tail; j++)
21         {
22             int tmp = que[j % n];
23             for (edge *i = g[tmp]; i != NULL; i = i->nxt)
24             {
25                 if (pool[OP(i)].cap > 0 && !vst[i->v])
26                 {
27                     tail++;
28                     vst[i->v] = 1;
29                     que[tail % n] = i->v;
30                     dis[i->v] = dis[tmp] + 1;
31                     numdis[dis[i->v]]++;
32                 }
33             }
34         }
35     }
36     int findArgumentPath(int &v, int s, int t)
37     {
38         while (e[v] != NULL)
39         {
40             if (e[v]->cap > 0 && dis[v] == dis[e[v]->v] + 1)
41             {
42                 pre[e[v]->v] = v, pree[e[v]->v] = e[v], v = e[v]->v;

```

```

43         if (v == t)
44         {
45             int minf = INF;
46             for (int i = t; i != s; i = pre[i])
47                 minf = min(minf, pree[i]->cap);
48             for (int i = t; i != s; i = pre[i])
49             {
50                 pree[i]->cap -= minf;
51                 pool[OP(pree[i])].cap += minf;
52             }
53             v = s;
54             return minf;
55         }
56     }
57     else e[v] = e[v]->nxt;
58 }
59 return 0;
60 }
61 public:
62 int maxflowsap(int n, int s, int t)
63 {
64     bfs(t, n);
65     int v = s;
66     copy(g, g + n, e);
67     while (dis[s] < n) //标号为n 表示无可行流
68     {
69         int add = findArgumentPath(v, s, t);
70         maxflow += add;
71         if (add == 0) //发现某个点没有允许弧, 维护其距离标号v
72         {
73             int mindis = n;
74             numdis[dis[v]]--;
75             if (!numdis[dis[v]]) break; //GAP 优化, 发现断层直接退出
76             for (edge *i = g[v]; i != NULL; i = i->nxt)
77                 if (i->cap > 0) mindis = min(mindis, dis[i->v] + 1);
78             dis[v] = mindis;
79             numdis[dis[v]]++;
80             e[v] = g[v]; //改变距离标号以后从新维护当前弧, 回前驱
81             if (v != s) v = pre[v];
82         }
83     }
84     return maxflow;
85 }
86 void firststart()
87 {
88     pp = pool;
89     maxflow = 0;
90     memset(g, 0, sizeof(g));
91     //memset(reachS, 0, sizeof(reachS));
92     //memset(reachT, 0, sizeof(reachT));
93 } //后两个用于求割等问题
94 void addedge(int i, int j, int cap)
95 {
96     pp->v = j;
97     pp->cap = cap;
98     pp->nxt = g[i];
99     g[i] = pp++;
100 } //不自动加反向边, 如果i to j j to i 都有容量且相邻加入i
101 void dfss(int x)
102 {
103     reachS[x] = true;
104     for (edge *i = g[x]; i != NULL; i = i->nxt)
105         if (i->cap && !reachS[i->v]) dfss(i->v);
106 } //网络流割S-割出来的集合是从源点正向边遍历到的点集合TS
107 void dfst(int x)
108 {

```



```

109     reachT[x] = true;
110     for(edge *i = g[x]; i != NULL; i = i->nxt)
111         if(pool[OP(i)].cap && !reachT[i->v]) dfst(i->v);
112 } //用于求关键边等问题
113 //关键边是一端reachS 一端reachT 且无流量的边
114 }maxflow;

```

3.25 mincostflow

```

1  using namespace std;
2  typedef long long USETYPE;
3  const USETYPE INF = numeric_limits<USETYPE>::max(); //<limits>
4  template<typename T = int>
5  class mincost
6  {
7  private:
8      const static int N = 1000;
9      const static int E = 100000;
10     struct edge
11     {
12         int u, v;
13         T cost, cap;
14         edge *nxt;
15     } pool[E], *g[N], *pp, *pree[N];
16     T dist[N];
17
18     bool SPFA(int n, int s, int t)
19     {
20         fill(dist, dist + n, INF);
21         int tail = 0, q[N] = {s};
22         dist[s] = 0;
23         bool vst[N] = {false};
24         vst[s] = true;
25         for(int i = 0; i <= tail; i++)
26         {
27             int u = q[i % n];
28             for(edge *j = g[u]; j != NULL; j = j->nxt)
29             {
30                 int v = j->v;
31                 if(j->cap && dist[u] != INF && dist[v] > dist[u] + j->cost)
32                 {
33                     dist[v] = dist[u] + j->cost;
34                     pree[v] = j;
35                     if(!vst[v])
36                     {
37                         tail++;
38                         q[tail % n] = v;
39                         vst[v] = true;
40                     }
41                 }
42             }
43             vst[u] = false;
44         }
45         return dist[t] < INF;
46     }
47 public:
48     #define OP(i) (((i) - pool) ^ 1)
49     void addedge(int u, int v, T cap, T cost)
50     {
51         pp->u = u, pp->v = v;
52         pp->cost = cost, pp->cap = cap;
53         pp->nxt = g[u], g[u] = pp++;
54     }
55     void initialize()

```

```

56     {
57         CC(g, 0);
58         pp = pool;
59     }
60     pair<T, T> mincostflow(int n, int s, int t)
61     {
62         T flow = 0, cost = 0;
63         while(SPFA(n, s, t))
64         {
65             T minf = INF;
66             for(int i = t; i != s; i = pre[i]->u)
67                 minf = min(minf, pre[i]->cap);
68             for(int i = t; i != s; i = pre[i]->u)
69             {
70                 pre[i]->cap -= minf;
71                 pool[OP(pre[i])].cap += minf;
72                 cost += minf * pre[i]->cost;
73             }
74             flow += minf;
75         }
76         return make_pair(flow, cost);
77     }
78 };

```

4 computational geometry

4.1 geometry

```
1  const double EPS = 1e-8;
2  const double PI = acos(-1.0);
3  const double INF = 1e100;
4  struct Point
5  {
6      double x, y;
7      Point(double xx = 0, double yy = 0)
8      {x = xx, y = yy;}
9      bool operator <(const Point a) const
10     {return y == a.y ? x < a.x : y < a.y;}
11     friend ostream& operator << (ostream& out, Point a)
12     {
13         out << "(" << a.x << " " << a.y << ")";
14         return out;
15     }
16 };
17 /* *****
18  * 距离公式
19  * *****/
20 double dist(double x1, double y1, double x2, double y2)
21 {
22     return sqrt(pow(x1 - x2, 2.0) + pow(y1 - y2, 2.0));
23 }
24 double sphereDist(double x1, double y1, double x2, double y2, double R = 1)
25 {
26     //longitude x and latitude y
27     //z[i] = sin(lat[i]*PI/180);
28     //x[i] = cos(lng[i]*PI/180) * cos(lat[i]*PI/180);
29     //y[i] = sin(lng[i]*PI/180) * cos(lat[i]*PI/180);
30     //dist = x[i]*x[j] + y[i]*y[j] + z[i]*z[j]
31     x1 /= 180; y1 /= 180; x2 /= 180; y2 /= 180;
32     x1 *= PI; y1 *= PI; x2 *= PI; y2 *= PI;
33     return R * acos(sin(y1) * sin(y2) + cos(y1) * cos(y2) * cos(x1 - x2));
34 }
35
36 /* *****
37  * 基础应用
38  * *****/
39 int dblcmp(double x)
40 {
41     if(fabs(x) < EPS) return 0;
42     return x < 0 ? -1: 1;
43 }
44 double det(double x1, double y1, double x2, double y2)
45 {
46     return x1 * y2 - x2 * y1;
47 }
48 double cross(Point a, Point b, Point c) //ab x ac
49 {
50     return det(b.x - a.x, b.y - a.y, c.x - a.x, c.y - a.y);
51 }
52 double dotbet(double x1, double y1, double x2, double y2)
53 {
54     return x1 * x2 + y1 * y2;
55 }
56 double dot(Point a, Point b, Point c)
57 {
58     return dotbet(b.x - a.x, b.y - a.y, c.x - a.x, c.y - a.y);
59 }
60 int betweencmp(Point a, Point b, Point c)
61 {
```

```

62     return dblcmp(dot(a, b, c));
63 }
64 /* *****
65 * 直线线段相交模板
66 * *****/
67 bool segcrosssimple(Point a, Point b, Point c, Point d)
68 { //ab 与是否规范相交cd
69     return (dblcmp(cross(a, c, d)) ^ dblcmp(cross(b, c, d))) == -2 &&
70            (dblcmp(cross(c, a, b)) ^ dblcmp(cross(d, a, b))) == -2;
71 }
72 int segcross(Point a, Point b, Point c, Point d, Point& p)
73 { //ab 是否相交, 规范相交返回交点cd
74     double s1, s2, s3, s4;
75     int d1 = dblcmp(s1 = cross(a, b, c));
76     int d2 = dblcmp(s2 = cross(a, b, d));
77     int d3 = dblcmp(s3 = cross(c, d, a));
78     int d4 = dblcmp(s4 = cross(c, d, b));
79     if((d1 ^ d2) == -2 && (d3 ^ d4) == -2)
80     {
81         p.x = (c.x * s2 - d.x * s1) / (s2 - s1);
82         p.y = (c.y * s2 - d.y * s1) / (s2 - s1);
83         return 1;
84     }
85     if(d1 == 0 && betweencmp(c, a, b) <= 0 ||
86        d2 == 0 && betweencmp(d, a, b) <= 0 ||
87        d3 == 0 && betweencmp(a, c, d) <= 0 ||
88        d4 == 0 && betweencmp(b, c, d) <= 0) return 2;
89     return 0;
90 }
91 int linecrossseg(Point a, Point b, Point c, Point d, Point& temp)
92 { //直线与线段相交, 返回相交交点abcd
93     double s1, s2;
94     int d1, d2;
95     d1 = dblcmp(s1 = cross(a, b, c));
96     d2 = dblcmp(s2 = cross(a, b, d));
97     if (d1 * d2 < 0) {
98         temp.x = (c.x * s2 - d.x * s1) / (s2 - s1);
99         temp.y = (c.y * s2 - d.y * s1) / (s2 - s1);
100        return 1;
101    }
102    if (d1 * d2 == 0) //交于端点
103    {
104        if(d2 == 0) temp = d;
105        else temp = c;
106        return 2;
107    }
108    return 0;
109 }
110 bool linecross(Point a, Point b, Point c, Point d, Point& temp)
111 { //直线与直线是否相交, 相交返回交点abcd
112     if((b.x - a.x) * (d.y - c.y) == (d.x - c.x) * (b.y - a.y)) return false;
113     double s1, s2;
114     int d1, d2;
115     d1 = dblcmp(s1 = cross(a, b, c));
116     d2 = dblcmp(s2 = cross(a, b, d));
117     temp.x = (c.x * s2 - d.x * s1) / (s2 - s1);
118     temp.y = (c.y * s2 - d.y * s1) / (s2 - s1);
119     return true;
120 }
121
122 /* *****
123 * 凸包模板
124 * *****/
125 void ConvexHull(Point* pts, Point* stk, int n, int &top) //p[0] != [n - 1]
126 {
127     sort(pts, pts + n);

```

```

128     top = -1;
129     stk[++top] = pts[0];
130     stk[++top] = pts[1];
131     for(int i = 2; i < n; i++)
132     {
133         while(top >= 1 && dblcmp(cross(stk[top - 1], stk[top], pts[i])) <=
            0)
134             top--;
135         stk[++top] = pts[i];
136     }
137     int now = top;
138     for(int i = n - 2; i >= 0; i--)
139     {
140         while(top >= now + 1 && dblcmp(cross(stk[top - 1], stk[top], pts[i]
            ))) <= 0)
141             top--;
142         stk[++top] = pts[i];
143     }
144 }
145 /* *****
146  * 旋转卡壳专用
147  * *****/
148 double rotating_calipers_longest(Point* p, int n) //卡壳
149 {
150     double res = 0;
151     p[n] = p[0];
152     for(int i = 0, j = 1; i < n; i++)
153     {
154         while(dblcmp(cross(p[i], p[i + 1], p[j]) - cross(p[i], p[i + 1], p
            [(j + 1) % n])) < 0)
155             j = (j + 1) % n;
156         res = max(res, fabs(cross(p[i], p[i + 1], p[j])));
157     }
158     return res;
159 }
160 double rotating_calipers_triangle(Point p[], int n)
161 {
162     int i, j = 1, q = 2;
163     p[n] = p[0];
164     p[n+1] = p[1];
165     p[n+2] = p[2];
166     double temp, ans = 0;
167     for (i = 0; i < n; i++)
168     {
169         while (cross(p[i], p[j], p[q+1]) - (temp = cross(p[i], p[j], p[q])) >
            EPS)
170             q = (q + 1) % n;
171         ans = max(ans, temp);
172         while (cross(p[i], p[j+1], p[q]) - (temp = cross(p[i], p[j], p[q])) >
            EPS)
173             j = (j + 1) % n;
174         ans = max(ans, temp);
175     }
176     return ans;
177 }
178
179 /* *****
180  * 多边形重心
181  * *****/
182 Point barycenter(Point a, Point b, Point c)
183 {
184     Point tmp;
185     linecross(Point((a.x + b.x) / 2, (a.y + b.y) / 2), c,
186         Point((a.x + c.x) / 2, (a.y + c.y) / 2), b, tmp);
187     return tmp;
188 }

```

```

189 Point barycenter(Point p[], int n)
190 {
191     Point ret, t;
192     double t1 = 0, t2;
193     ret.x = ret.y = 0;
194     for (int i = 1; i < n - 1; i++)
195         if (fabs(t2 = cross(p[0], p[i], p[i + 1])) > EPS)
196         {
197             t = barycenter(p[0], p[i], p[i + 1]);
198             ret.x += t.x * t2;
199             ret.y += t.y * t2;
200             t1 += t2;
201         }
202     if (fabs(t1) > EPS)
203         ret.x /= t1, ret.y /= t1;
204     return ret;
205 }
206 Point verticalfoot(Point a, Point b, Point c)
207 { //在上都的垂足cab
208     Point tmp(c.x - a.y + b.y, c.y + a.x - b.x), ans;
209     linecross(a, b, c, tmp, ans);
210     return ans;
211 }

```

4.2 3D-Convex

```

1  #include <iostream>
2  #include <cstring>
3  #include <cstdio>
4  #include <cmath>
5  #include <stdlib.h>
6  #include <vector>
7  using namespace std;
8  int faces;
9  int sig(double x)
10 {
11     return (x > 1E-6) - (x < -1E-6);
12 }
13 #define N 505
14 struct Point
15 {
16     double x, y, z;
17     Point() {}
18     Point(double x, double y, double z) : x(x), y(y), z(z) {}
19     Point operator +(Point b)
20     {
21         return Point(x + b.x, y + b.y, z + b.z);
22     }
23     Point operator -(Point b)
24     {
25         return Point(x - b.x, y - b.y, z - b.z);
26     }
27     Point operator /(double t)
28     {
29         return Point(x / t, y / t, z / t);
30     }
31     double len()
32     {
33         return sqrt(x * x + y * y + z * z);
34     }
35 };
36 double dot(Point a, Point b)
37 {
38     return a.x * b.x + a.y * b.y + a.z * b.z;

```

```

39 }
40 Point cross(Point a, Point b)
41 {
42     return Point(a.y * b.z - a.z * b.y,
43                 -(a.x * b.z - a.z * b.x),
44                 a.x * b.y - a.y * b.x);
45 }
46 Point ps[N];
47 struct Face
48 {
49     int a,b,c;
50     Face(int a ,int b , int c ): a (a), b(b) , c(c) {}
51     double area ()
52     {
53         return cross( ps[b]-ps[a] , ps[c]-ps[a] ).len();
54     }
55     Point fa() const
56     {
57         return cross( ps[b]-ps[a] , ps[c]-ps[a] );
58     }
59     bool same_side(Point q , Point p)
60     {
61         return sig ( dot(ps[a] - q, cross(ps[b] - q, ps[c] - q))
62                     * dot(ps[a] - p , cross( ps[b] - p , ps[c] - p)) ) > 0 ;
63     }
64     bool inFace(Point q) const
65     {
66         return sig(dot(ps[a] - q, cross(ps[b] - q, ps[c] - q)))==0;
67     }
68     bool operator == (const Face & face) const
69     {
70         Point fa1 = fa();
71         Point fa2 = face.fa();
72         if(sig(cross(fa1,fa2).len())!=0) return false;
73         return inFace(ps[face.a]);
74     }
75 };
76 struct line
77 {
78     int a, b;
79     line(int a, int b) : a(a),b(b) {}
80 };
81 double convexHull(Point *ps, int n)
82 {
83     #define judge(S, T) \
84     map[C[j].S][C[j].T]=map[C[j].T][C[j].S]= map[C[j].S][C[j].T]==0;\
85     LT.push_back(line(C[j].S, C[j].T))
86     static bool map[N][N];
87     static vector <Face> C , FT;
88     static vector <line> LT;
89     int i, j;
90     if(n <= 2) return 0.0;
91     if(n == 3) return cross(ps[1]-ps[0] , ps[2]-ps[0]).len()*0.5;
92     C.clear();
93     memset(map, 0 , sizeof(map));
94     for(i = 0; i < 4; i ++)
95         C.push_back(Face(i, (i+1)%4, (i+2)%4));
96     Point center = (ps[0] + ps[1] + ps[2] + ps[3]) / 4;
97     for(i = 4 ; i < n ; i ++ )
98     {
99         FT.clear();
100        LT.clear();
101        for (j = 0 ; j < C.size() ; j ++ )
102            if ( ! (C[j].same_side( center , ps[i] )) )
103            {
104                judge(a, b);

```

```

105         judge(c, b);
106         judge(c, a);
107     }
108     else FT.push_back(C[j]);
109     C.clear();
110     for(j = 0 ; j < FT.size() ; j ++ )
111         C.push_back( FT[j] );
112     for(j = 0 ; j < LT.size() ; j ++ )
113         if (map [ LT[j].a ][ LT[j].b ])
114         {
115             C.push_back( Face ( LT[j].a , LT[j].b , i ) );
116             map[ LT[j].a ][ LT[j].b ] = map[ LT[j].b ][ LT[j].a ] = 0;
117         }
118     }
119     double area = 0 ;
120     for ( i = 0 ; i < C.size() ; i ++ )
121         area += C[i].area();
122     area /= 2.0;
123
124     faces = 0;
125     for(int i = 0; i < C.size(); i ++)
126     {
127         bool ok = true;
128         for(int j = i+1; j < C.size(); j ++)
129         {
130             if(C[i]==C[j])
131             {
132                 ok = false;
133                 break;
134             }
135         }
136         faces += ok;
137     }
138     return area;
139 }
140 int main()
141 {
142     int n;
143     double x, y, z;
144     while(scanf("%d", &n) != EOF)
145     {
146         for(int i = 0; i < n; i ++)
147         {
148             scanf("%lf%lf%lf", &x, &y, &z);
149             ps[i] = Point(x, y, z);
150         }
151         while(sig(convexHull(ps, n)) == 0)
152         {
153             for(int j = n-1; j > 0; j --)
154             {
155                 swap(ps[j], ps[rand()%j]);
156             }
157         }
158         printf("%d\n", faces);
159     }
160     return 0;
161 }

```

5 math

5.1 cantor extend

```
1 #include <iostream>
2 #include <cstring>
3 #include <cstdio>
4 using namespace std;
5 const int INF = 0xFFFFFFFF;
6 const int FAC_N = 10; // 0! to (n - 1)!
7 int fac[FAC_N] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};
8 int cantor(int *a, int n)
9 {
10     int ans = 0, i, j, r;
11     char p[10] = {0};
12     for (i = 0; i < n; i++)
13     {
14         for (j = 1, r = 0; j <= a[i]; j++)
15             if (p[j] == 0) r++;
16         ans += (r - 1) * fac[8 - i];
17         p[a[i]] = 1;
18     }
19     return ans;
20 }
21
22 void uncantor(int s, int *a, int n) // 0 to n - 1
23 {
24     int i, j, r, t;
25     char p[FAC_N] = {0};
26     for (i = 0; i < n; i++)
27     {
28         t = s / fac[n - 1 - i] + 1;
29         s %= fac[n - 1 - i];
30         r = 0, j = 1;
31         while (1)
32         {
33             if (p[j] == 0) r++;
34             if (r == t) break;
35             j++;
36         }
37         a[i] = j;
38         p[j] = 1;
39     }
40     return;
41 }
```

5.2 euler function

```
1 //f(n * m) = f(n) * f(m)
2 //f(n) = n * (1 - 1 / p1) * (1 - 1 / p2) (piis n's prim factor)
3 int euler(int n)
4 {
5     int e;
6     int i, j;
7     e = n;
8     for (i = 2; i * i <= n; i++)
9     {
10         if (n % i == 0)
11         {
12             e = e / i * (i - 1);
13             while (n % i == 0)
14                 n = n / i;
15         }
16     }
```

```

17     if (n > 1)
18         e = e / n * (n - 1);
19     return e;
20 }

```

5.3 Matrix

```

1  #define MOD 100000000
2  using namespace std;
3  const int maxn = 5;
4  struct Matrix
5  {
6      long long A[maxn][maxn];
7      int size;
8
9      Matrix()
10     {
11         memset(this, 0, sizeof (*this));
12     }
13 };
14 long long mymod(long long x)
15 {
16     return (x % MOD + MOD) % MOD;
17 }
18 Matrix operator+(Matrix m1, Matrix m2)
19 {
20     Matrix ret;
21     ret.size = m1.size;
22     for (int i = 0; i < ret.size; ++i)
23         for (int j = 0; j < ret.size; ++j)
24             ret.A[i][j] = mymod(m1.A[i][j] + m2.A[i][j]);
25     return ret;
26 }
27 Matrix operator-(Matrix m1, Matrix m2)
28 {
29     Matrix ret;
30     ret.size = m1.size;
31     for (int i = 0; i < ret.size; ++i)
32         for (int j = 0; j < ret.size; ++j)
33             ret.A[i][j] = mymod(m1.A[i][j] - m2.A[i][j]);
34     return ret;
35 }
36 Matrix operator*(Matrix m1, Matrix m2)
37 {
38     Matrix ret;
39     ret.size = m1.size;
40     for (int i = 0; i < ret.size; ++i)
41         for (int j = 0; j < ret.size; ++j)
42         {
43             ret.A[i][j] = 0;
44             for (int k = 0; k < ret.size; ++k)
45                 ret.A[i][j] += m1.A[i][k] * m2.A[k][j];
46             ret.A[i][j] = mymod(ret.A[i][j]);
47         }
48     return ret;
49 }
50 Matrix mypower(Matrix m, int n)
51 {
52     Matrix ret, tmp;
53     ret.size = m.size;
54     if (n == 0)
55     {
56         for (int i = 0; i < ret.size; ++i)
57             ret.A[i][i] = 1;

```

```

58     return ret;
59 }
60 tmp = mypower(m, n / 2);
61 if (n & 1)
62     return tmp * tmp * m;
63 else return tmp * tmp;
64 }
65 Matrix sumpower(Matrix m, int n)
66 {
67     Matrix tmp;
68     if (n == 1) return m;
69     tmp = sumpower(m, n / 2);
70     if (n & 1)
71         return mypower(m, n / 2) * tmp + tmp + mypower(m, n);
72     return mypower(m, n / 2) * tmp + tmp;
73 }

```

5.4 miller rabin

```

1  /*miller-rabin algorithm do it 5 times or more*/
2  long long qmod(long long a,long long b,long long c)
3  {
4      long long res = 1,temp = a % c;
5      while(b)
6      {
7          if(b & 1)
8              res = (res * temp) % c;
9          b>>= 1;
10         temp = (temp * temp) % c;
11     }
12     return res;
13 }
14
15 bool miller(long long n,int t)
16 {
17     if(n == 1)
18         return false;
19     else if(n == 2)
20         return true;
21     for(int i = 0;i < t;i++)
22     {
23         srand(time(NULL));
24         long long a = rand() % (n - 2) + 1;
25         int b = qmod(a,n - 1,n);
26         if(b != 1 && b != n - 1)
27             return false;
28     }
29     return true;
30 }
31
32 int main()
33 {
34     long long a;
35     while(scanf("%lld",&a) == 1)
36     {
37         if(miller(a,4))
38             printf("YES\n");
39         else
40             printf("NO\n");
41     }
42     return 0;
43 }

```

5.5 pollard rho

```
1 typedef long long LL;
2 LL min;
3 LL multi(LL a, LL b, LL n)
4 {
5     LL tmp = a % n, s = 0;
6     while(b)
7     {
8         if(b & 1) s = (s + tmp) % n;
9         tmp = (tmp + tmp) % n;
10        b >>= 1;
11    }
12    return s;
13 }
14 LL gcd(LL a, LL b)
15 {
16     return b ? gcd(b, a % b) : a;
17 }
18 LL pollard_rho(LL n, LL c)
19 {
20     LL x, y, d, i = 1, k = 2;
21     srand((LL) (0));
22     x = ((LL) rand()) % (n - 1) + 1;
23     y = x;
24     while(1)
25     {
26         i++;
27         x = (multi(x, x, n) + c) % n;
28         d = gcd(y - x + n, n);
29         if(d != 1 && d != n) return d;
30         if(y == x) return n;
31         if(i == k) y = x, k <<= 1;
32     }
33 }
34 void find(LL n, LL c)
35 {
36     LL r;
37     if(n <= 1) return;
38     if(test(n))
39     {
40         if(min > n) min = n;
41         return;
42     }
43     r = pollard_rho(n, c--);
44     find(n / r, c);
45     find(r, c);
46 }
47 LL MaxPrimeFactor(LL n)
48 {
49     if(test(n)) return n;
50     LL k = -1, g;
51     min = n;
52     find(n, C);
53     g = MaxPrimeFactor(min);
54     k = g > k ? g : k;
55     g = MaxPrimeFactor(n / min);
56     k = g > k ? g : k;
57     return k;
58 }
59 int main()
60 {
61     LL n;
62     while(scanf("%lld", &n) == 1)
63     {
```

```

64     if(test(n)) //test(n) is miller robin
65         printf("Yes\n");
66     else
67     {
68         min = n; //min is the min factor of n
69         find(n, C);
70         printf("No %lld\n",min);
71         //printf("%lld\n",MaxPrimeFactor(n));
72     }
73 }
74 return 0;
75 }

```

5.6 linearModularSystem

```

1  typedef long long LL;
2  LL gcd(LL a, LL b)
3  {
4      if(b == 0) return a;
5      return gcd(b, a % b);
6  }
7  LL extended_euclid(LL a,LL b,LL &x,LL &y)
8  {
9      if (b == 0)
10     {
11         x = 1, y = 0;
12         return a;
13     }
14     LL ret = extended_euclid(b ,a % b, x, y), t = x;
15     x = y;
16     y = t - a / b * y;
17     return ret;
18 }
19 bool modular_linear(LL a,LL b,LL c)
20 {
21     LL x, y;
22     LL d = extended_euclid(a,b,x,y);
23     if (c%d) return 0;
24     return true;
25 }
26 LL linearModularSystem(LL* m, LL* r, int n)//保证互质m且有节,
27 {
28     LL M = accumulate(m, m + n, 1, multiplies<LL>());
29     LL ans = 0;
30     for(int i = 0; i < n; i++)
31     {
32         LL Mi = M / m[i], pi, qi;
33         LL gcd = extended_euclid(Mi, m[i], pi, qi);
34         if(Mi % gcd) return -1;
35         ans = (ans + Mi * pi * r[i]) % M;
36     }
37     return ans <= 0 ? ans + M : ans;
38 }//minimum non-negative answer
39 LL linearModularSystemP(LL* m, LL* r, int n)//不互质
40 {
41     LL m0 = m[0], r0 = r[0];//前一方程
42     LL m1, r1;//当前方程
43     LL x, y, t;
44     for(int i = 1; i < n; i++)
45     {
46         r1 = r[i], m1 = m[i];
47         long long gcd = extended_euclid(m0, m1, x, y);
48         LL c = r1 - r0;
49         if(c % gcd != 0) return -1;

```

```

50         //m0 * x - m1 * y = r1 - r0
51         t = m1 / gcd; //倍数
52         x = (c / gcd * x % t + t) % t; //最小正整数解
53         r0 = r0 + x * m0;
54         m0 = m0 * m1 / gcd;
55     }
56     return r0;
57 }

```

5.7 Eratosthenes

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  using namespace std;
5  const int N = 100000;
6  int tag[N], p[N];
7  void get_prime()
8  {
9      int cnt = 0;
10     for (int i = 2; i < N; i++)
11     {
12         if (!tag[i]) p[cnt++] = i;
13         for (int j = 0; j < cnt && p[j] * i < N; j++)
14         {
15             tag[i*p[j]] = 1;
16             if (i % p[j] == 0) break;
17         }
18     }
19 }

```

6 dynamic Programming

6.1 DFA DP

```
1 LL DP(LL a, int k)
2 {
3     char str[N * 2];
4     sprintf(str, "%lld", a);
5     LL dp[2][2][N][N][N], ans = 0; //section, 0\1, length, mod
6     int len = strlen(str);
7     memset(dp, 0, sizeof(dp)); //deal with prefix 0
8     dp[0][1][0][(str[0] - '0') % k][0]++;
9     for(int now = 1; now < str[0] - '0'; now++)
10         dp[0][0][0][now % k][0]++;
11     REP(nxt, 1, len)
12     {
13         REP(now, 1, 10) //deal with prefix 0
14             dp[0][0][nxt][now % k][0]++;
15         REP(a, 0, k)
16         {
17             REP(b, 0, k)
18             {
19                 for(int now = 0; now < 10; now++)
20                 {
21                     if(now == str[nxt] - '0')
22                     {
23                         dp[0][1][nxt][(a * 10 + now) % k][b] += dp[0][1][nxt-1][a][b];
24                         dp[1][1][nxt][a][(b * 10 + now) % k] += dp[0][1][nxt-1][a][b];
25                         dp[1][1][nxt][a][(b * 10 + now) % k] += dp[1][1][nxt-1][a][b];
26                     }
27                     else if(now < str[nxt] - '0')
28                     {
29                         dp[0][0][nxt][(a * 10 + now) % k][b] += dp[0][1][nxt-1][a][b];
30                         dp[1][0][nxt][a][(b * 10 + now) % k] += dp[0][1][nxt-1][a][b];
31                         dp[1][0][nxt][a][(b * 10 + now) % k] += dp[1][1][nxt-1][a][b];
32                     }
33                     dp[0][0][nxt][(a * 10 + now) % k][b] += dp[0][0][nxt-1][a][b];
34                     dp[1][0][nxt][a][(b * 10 + now) % k] += dp[0][0][nxt-1][a][b];
35                     dp[1][0][nxt][a][(b * 10 + now) % k] += dp[1][0][nxt-1][a][b];
36                 }
37             }
38         }
39     }
40     for(int a = 0; a < k; a++)
41         for(int b = 0; b < k; b++)
42             if((a + b) % k == 0)
43                 ans += dp[1][0][len-1][a][b] + dp[1][1][len-1][a][b];
44     return ans;
45 }
46 int main ()
47 {
48     LL a, b;
49     int k;
50     while (scanf ("%lld%lld%d", &a, &b, &k) == 3)
51         printf("%lld\n", DP(b, k) - DP(a-1, k));
52     return 0;
53 }
```

6.2 mask & connection

```
1 typedef long long LL;
2 const int N = 14;
3 const int TOT = 50000;
4 const int MAXN = 1594323; // 3^13
```

```

5 char maps[N][N];
6 int bit3[N] = {1}, status[TOT];
7 int Hash[MAXN], allS = 0;
8 LL dp[2][TOT];
9 bool check(int s)
10 {
11     int cnt = 0;
12     while(s)
13     {
14         int n = s % 3;
15         if(n == 1) cnt++;
16         if(n == 2) cnt--;
17         if(cnt < 0) return false;
18         s /= 3;
19     }
20     return (cnt == 0);
21 }
22 void preprocess()
23 {
24     REP(i, 1, N) bit3[i] = bit3[i - 1] * 3;
25     REP(i, 0, bit3[N - 1])
26     {
27         if(check(i))
28         {
29             Hash[i] = allS;
30             status[allS++] = i;
31         }
32         else Hash[i] = -1;
33     }
34     status[allS] = MAXN;
35 }
36 int getbit(int s, int i)
37 {
38     while(i-- > 0) s /= 3;
39     return s % 3;
40 }
41 void transfer(LL& dest, LL add)
42 {
43     dest == -1 ? (dest = add) : (dest += add);
44 }
45 LL DP(int n, int m, int px, int py)
46 {
47     LL ans = 0;
48     int now = 0, pre;
49     CC(dp, -1);
50     dp[0][0] = 1;
51     for(int i = 0; i < n; i++)
52     {
53         for(int j = 0; j < m; j++)
54         {
55             pre = now; now ^= 1;
56             CC(dp[now], -1);
57             for(int k = 0, s; s = status[k], s < bit3[m + 1]; k++)
58             {
59                 if(dp[pre][k] == -1) continue;
60                 int l = getbit(s, j), u = getbit(s, j + 1);
61                 int nows = s - l * bit3[j] - u * bit3[j + 1];
62                 if(maps[i][j] == '*')
63                 {
64                     if(l == 0 && u == 0)
65                         transfer(dp[now][k], dp[pre][k]);
66                 }
67                 else if(l == 0 && u == 0) //both down and right build 2 plugin
68                 {
69                     if(maps[i][j + 1] == '.' && maps[i + 1][j] == '.')
70                     {

```



```

71         int nxt = nows + bit3[j] + 2 * bit3[j + 1];
72         transfer(dp[now][Hash[nxt]], dp[pre][k]);
73     }
74 }
75 else if(l == 1 && u == 1)// merge (( make )) to ()
76 {
77     int cnt = 0;
78     for(int b = j + 2; b <= m; b++)
79     {
80         int tmp = getbit(nows, b);
81         if(tmp == 2) cnt--;
82         if(tmp == 1) cnt++;
83         if(cnt == -1)
84         {
85             transfer(dp[now][Hash[nows - bit3[b]]], dp[pre][k]);
86             if(Hash[nows - bit3[b]] == -1)
87                 cout << nows - bit3[b] << endl;
88             break;
89         }
90     }
91 }
92 else if(l == 2 && u == 2)// merge )) make (( to ()
93 {
94     int cnt = 0;
95     for(int b = j - 1; b >= 0; b--)
96     {
97         int tmp = getbit(nows, b);
98         if(tmp == 1) cnt++;
99         if(tmp == 2) cnt--;
100        if(cnt == 1)
101        {
102            transfer(dp[now][Hash[nows + bit3[b]]], dp[pre][k]);
103            if(Hash[nows + bit3[b]] == -1)
104                cout << nows + bit3[b] << endl;
105            break;
106        }
107    }
108 }
109 else if(l == 1 && u == 2)//merge () at last grid
110 {
111     if(px == i && py == j)
112         ans += dp[pre][k];
113 }
114 else if(l == 2 && u == 1)//merge )(
115 {
116     transfer(dp[now][Hash[nows]], dp[pre][k]);
117 }
118 else if((!l && u) || (l && !u))
119 {
120     if(maps[i + 1][j] == '.')
121         transfer(dp[now][Hash[nows + (l + u) * bit3[j]]], dp[pre][k])
122         ;
123     if(maps[i][j + 1] == '.')
124         transfer(dp[now][Hash[nows + (l + u) * bit3[j + 1]]], dp[pre
125 ] [k]);
126 }
127 }
128 pre = now; now ^= 1;
129 CC(dp[now], -1); //must CC -1
130 for(int k = 0, s; s = status[k], s < bit3[m]; k++)
131     if(dp[pre][k] != -1)
132         dp[now][Hash[s * 3]] = dp[pre][k];
133 }
134 return ans;
135 }

```

```

135 int main()
136 {
137     int n, m, px, py;
138     preprocess();
139     while (scanf("%d %d", &n, &m) == 2)
140     {
141         CC(maps, 0);
142         REP(i, 0, n) scanf("%s", maps[i]);
143         REP(i, 0, n) REP(j, 0, m) if (maps[i][j] == '.') px = i, py = j;
144         printf("%lld\n", DP(n, m, px, py));
145     }
146     return 0;
147 }

```

6.3 RMQ

```

1  /*RMQ-ST be careful with log2*/
2  const int SIZE = 500001;
3  int dp[SIZE][20];
4  int n, q, a, b, len, ans;
5  int main()
6  {
7      while (scanf("%d %d", &n, &q) == 2)
8      {
9          for (int i = 1; i <= n; i++)
10             scanf("%d", &dp[i][0]);
11          for (int j = 1; j <= log(n) / log(2); j++)
12             for (int i = 1; i + (1 << (j - 1)) <= n; i++)
13                 dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
14          while (q--)
15          {
16              scanf("%d %d", &a, &b);
17              len = log(b - a + 1) / log(2) + 0.001;
18              ans = max(dp[a][len], dp[b - (1 << len) + 1][len]);
19              printf("%d\n", ans);
20          }
21      }
22      return 0;
23 }

```

6.4 HOJ2973

```

1  char from[201], to[201];
2  int dp[201][201][3];
3
4  int getmin(int start, int end, int state)
5  {
6      if (start > end)
7          return 0;
8      if (dp[start][end][state] < 100000)
9          return dp[start][end][state];
10     if ((from[start] == to[start] && state == 0) || state == to[start] - 'A' + 1)
11         dp[start][end][state] = getmin(start + 1, end, state);
12     else
13     {
14         for (int i = start; i <= end; ++i)
15             dp[start][end][state] = min(dp[start][end][state],
16             getmin(start, i, to[start] - 'A' + 1) + getmin(i + 1, end, state) + 1);
17     }
18     return dp[start][end][state];
19 }
20

```

```

21 int main()
22 {
23     int test;
24     scanf("%d", &test);
25     while (test--)
26     {
27         scanf("%s %s", from, to);
28         int len = strlen(from);
29         for (int i = 0; i < len; ++i)
30             for (int j = i; j < len; ++j)
31                 dp[i][j][0] = dp[i][j][1] = dp[i][j][2] = 100000;
32         printf("%d\n", getmin(0, len - 1, 0));
33     }
34     return 0;
35 }

```

6.5 slope

```

1  #include <iostream>
2  #include <cstring>
3  #include <cstdio>
4  using namespace std;
5
6  const int SIZE = 20002;
7  const LL INF = 2000000000LL;
8  long long w[SIZE], d[SIZE]; //input data
9  long long ds[SIZE], dp[SIZE];
10 long long sw[SIZE], sp[SIZE], bp[SIZE];
11 int n, q[SIZE], head, tail;
12
13 void preprocess()
14 {
15     ds[1] = 0;
16     for(int i = 2; i <= n + 1; i++)
17         ds[i] = ds[i - 1] + d[i - 1];
18     sw[0] = sp[0] = 0;
19     for(int i = 1; i <= n + 1; i++)
20     {
21         sw[i] = sw[i - 1] + w[i];
22         sp[i] = sp[i - 1] + sw[i - 1] * d[i - 1];
23     }
24     bp[n + 1] = 0;
25     for(int i = n; i >= 1; i--)
26         bp[i] = bp[i + 1] + w[i] * (ds[n + 1] - ds[i]);
27 }
28
29 void DP()
30 {
31     fill(dp + 1, dp + n + 1, INF);
32     long long ans = INF;
33     head = tail = 0;
34     q[tail++] = 1;
35     ans = min(ans, sp[n + 1] - sw[0] * (ds[1] - ds[0]) - sw[1] * (ds[n + 1] - ds[1]));
36     for(int i = 2; i <= n; i++)
37     {
38         /*
39         * bp[i + 1] = sp[n + 1] - sp[i] - sw[i] * (ds[n + 1] - ds[i]);
40         * dp[i] = min(dp[i], bp[i + 1] + sp[j] + sp[i] - sp[j] - sw[j] * (ds[i] - ds[j]));
41         * dp[i] = min(dp[i], sp[n + 1] - sw[j] * (ds[i] - ds[j]) - sw[i] * (ds[n + 1] - ds[i]));
42         * dp[i]k is the minimum index in 0-k dp[i]j - dp[i]k >= 0 (j < k)

```

```

43     * (sw[j] * ds[j] - sw[k] * ds[k]) / (sw[j] - sw[k]) <= ds[i] <= ds[i +
44         1];
45     * so i + 1 decision is more than k
46     * y = sw[j] * ds[j] x = sw[j];
47     */
48     while(head + 1 < tail)
49     {
50         long long y1 = sw[q[head]] * ds[q[head]], y2 = sw[q[head + 1]] * ds[q[
51             head + 1]];
52         long long x1 = sw[q[head]], x2 = sw[q[head + 1]];
53         if((y2 - y1) <= ds[i] * (x2 - x1)) head++;
54         else break;
55     }
56     int k = q[head];
57     dp[i] = sp[n + 1] - sw[k] * (ds[i] - ds[k]) - sw[i] * (ds[n + 1] - ds[i])
58         ;
59     ans = min(ans, dp[i]);
60     while(head + 1 < tail)
61     {
62         long long y1 = sw[q[tail - 2]] * ds[q[tail - 2]];
63         long long y2 = sw[q[tail - 1]] * ds[q[tail - 1]], y3 = sw[i] * ds[i];
64         long long x1 = sw[q[tail - 2]], x2 = sw[q[tail - 1]], x3 = sw[i];
65         if((y2 - y1) * (x3 - x2) >= (y3 - y2) * (x2 - x1)) tail--;
66         else break;
67     }
68     q[tail++] = i;
69     printf("%lld\n", ans);
70 }
71
72 int main()
73 {
74     while(scanf("%d", &n) == 1)
75     {
76         for(int i = 1; i <= n; i++)
77             scanf("%lld %lld", &w[i], &d[i]);
78         preprocess();
79         DP();
80     }
81     return 0;
82 }

```

7 other

7.1 连通性DP

```
1  class HashTable
2  {
3  private:
4      const static int SIZE = 1000000;
5      const static int MOD = 10007;
6      struct HashCell
7      {
8          int value, idx;
9          HashCell *nxt;
10     } pool[SIZE], *g[MOD], *pp;
11 #define hashFunction(x) ((x) % MOD)
12 public:
13     void clear()
14     {
15         memset(g, 0, sizeof(g));
16         pp = pool;
17     }
18     int find(int x)
19     {
20         int hash = hashFunction(x);
21         for(HashCell *i = g[hash]; i != NULL; i = i->nxt)
22         {
23             if(i->value == x)
24                 return i->idx;
25         }
26         return -1;
27     }
28     void insert(int x, int idx)
29     {
30         int hash = hashFunction(x);
31         pp->idx = idx;
32         pp->value = x;
33         pp->nxt = g[hash];
34         g[hash] = pp++;
35     }
36 } hashTable;
37 const int N = 10;
38 const int STATE_CNT = 1000000;
39 const int INF = 10000000;
40 const int HEX = 10;
41 const int BIT[] = {1, 10, 100, 1000, 10000, 100000,
42     1000000, 10000000, 100000000, 1000000000};
43 int state[2][STATE_CNT], dp[2][STATE_CNT];
44 int newState(int s[], const int M)
45 {
46     int lab[N], cnt = 0, newS = 0;
47     memset(lab, -1, sizeof(lab));
48     lab[0] = cnt++;
49     for(int i = M - 1; i >= 0; i--)
50     {
51         newS *= 10;
52         if(lab[s[i]] == -1)
53             lab[s[i]] = cnt++;
54         newS += lab[s[i]];
55     }
56     return newS;
57 }
58 int cnt[N];
59 void change(int src, int* dest, const int M)
60 {
61     memset(cnt, 0, sizeof(cnt));
```

```

62     REP(i, 0, M)
63     {
64         cnt[src % HEX]++;
65         dest[i] = src % HEX;
66         src /= HEX;
67     }
68 }
69 bool isOneBlock(int s)
70 {
71     int last = -1;
72     while(s)
73     {
74         int now = s % HEX;
75         if(now != 0 && now != last && last != -1) return false;
76         if(now != 0) last = now;
77         s /= HEX;
78     }
79     return true;
80 }
81 void transfer(int now, int newS, int val, int& newCnt)
82 {
83     int idx = hashTable.find(newS);
84     if(idx != -1)
85         dp[now][idx] = max(dp[now][idx], val);
86     else
87     {
88         idx = newCnt;
89         hashTable.insert(newS, newCnt++);
90         state[now][idx] = newS;
91         dp[now][idx] = val;
92     }
93 }
94 int DP(int n, int m, int maps[N][N])
95 {
96     int ans = -INF;
97     state[0][0] = dp[0][0] = 0;
98     int newS[N], now = 0, pre = 1;
99     int oldCnt, newCnt = 1;
100    REP(i, 0, n)
101    {
102        REP(j, 0, m)
103        {
104            now ^= 1, pre ^= 1;
105            oldCnt = newCnt;
106            newCnt = 0;
107            hashTable.clear();
108            REP(k, 0, oldCnt)
109            {
110                /*注意处理不选则该块的情况，如果不选择该块会导致一个联通块在表示法中消失。则改选
111                 法将导致多个联通块
112
113                 */
114                change(state[pre][k], newS, m + 1);
115                if(newS[j] == 0 && newS[j + 1] == 0)
116                {
117                    transfer(now, state[pre][k], dp[pre][k], newCnt);
118                    int minn = *max_element(newS, newS + m + 1) + 1;
119                    newS[j] = newS[j + 1] = minn;
120                    int nxt = newState(newS, m + 1);
121                    transfer(now, nxt, dp[pre][k] + maps[i][j], newCnt);
122                }
123                else if(newS[j] == 0 && newS[j + 1])
124                {
125                    newS[j] = newS[j + 1];
126                    int nxt = newState(newS, m + 1);
127                    transfer(now, nxt, dp[pre][k] + maps[i][j], newCnt);

```

```

127         if(cnt[newS[j + 1]] > 1)
128         {
129             newS[j] = newS[j + 1] = 0;
130             nxt = newState(newS, m + 1);
131             transfer(now, nxt, dp[pre][k], newCnt);
132         }
133     }
134     else if(newS[j] && newS[j + 1] == 0)
135     {
136         newS[j + 1] = newS[j];
137         int nxt = newState(newS, m + 1);
138         transfer(now, nxt, dp[pre][k] + maps[i][j], newCnt);
139         if(cnt[newS[j]] > 1)
140         {
141             newS[j] = newS[j + 1] = 0;
142             nxt = newState(newS, m + 1);
143             transfer(now, nxt, dp[pre][k], newCnt);
144         }
145     }
146     else if(newS[j] && newS[j + 1])
147     {
148         if(newS[j] == newS[j + 1])
149         {
150             if(cnt[newS[j]] > 2)
151             {
152                 int a = newS[j], b = newS[j + 1];
153                 newS[j] = newS[j + 1] = 0;
154                 int nxt = newState(newS, m + 1);
155                 transfer(now, nxt, dp[pre][k], newCnt);
156                 newS[j] = a, newS[j + 1] = b;
157             }
158         }
159         else
160         {
161             if(cnt[newS[j]] > 1 && cnt[newS[j + 1]] > 1)
162             {
163                 int a = newS[j], b = newS[j + 1];
164                 newS[j] = newS[j + 1] = 0;
165                 int nxt = newState(newS, m + 1);
166                 transfer(now, nxt, dp[pre][k], newCnt);
167                 newS[j] = a, newS[j + 1] = b;
168             }
169         }
170         int minn = min(newS[j], newS[j + 1]);
171         for(int b = 0; b <= m; b++)
172             if(newS[b] == newS[j] || newS[b] == newS[j + 1])
173                 newS[b] = minn;
174         int nxt = newState(newS, m + 1);
175         transfer(now, nxt, dp[pre][k] + maps[i][j], newCnt);
176     }
177 }
178 }
179 now ^= 1, pre ^= 1;
180 oldCnt = newCnt;
181 newCnt = 0;
182 hashTable.clear();
183 REP(k, 0, oldCnt)
184 {
185     if(isOneBlock(state[pre][k]))
186         ans = max(ans, dp[pre][k]);
187     if(state[pre][k] - BIT[m] > 0)
188     {
189         change((state[pre][k] - BIT[m]) * 10, newS, m + 1);
190         int nxt = newState(newS, m + 1);
191         if(nxt != 0)
192             transfer(now, nxt, dp[pre][k], newCnt);

```

```

193     }
194     else if(state[pre][k] != 0)
195     {
196         change(state[pre][k] * 10, newS, m + 1);
197         int nxt = newState(newS, m + 1);
198         if(nxt != 0)
199             transfer(now, nxt, dp[pre][k], newCnt);
200     }
201 }
202 transfer(now, 0, 0, newCnt);
203 }
204 return ans;
205 }
206
207 int main()
208 {
209     int n, m;
210     while(scanf("%d", &n) == 1 && n)
211     {
212         int maps[N][N], ans = -INF;
213         m = n;
214         REP(i, 0, n)
215         {
216             REP(j, 0, m)
217             {
218                 scanf("%d", &maps[i][j]);
219                 ans = max(ans, maps[i][j]);
220             }
221         }
222         if(ans <= 0)
223             printf("%d\n", ans);
224         else
225             printf("%d\n", DP(n, m, maps));
226     }
227     return 0;
228 }

```

7.2 input stream

```

1  class BufferedReader
2  {
3  public:
4      BufferedReader& operator >> (int& number)
5      {
6          number = getInt();
7          return *this;
8      }
9  private:
10     int getInt()
11     {
12         char ch;
13         while((ch = getchar()) && !isdigit(ch));
14         int ret = ch - '0';
15         while((ch = getchar()) && isdigit(ch))
16         {
17             ret *= 10;
18             ret += ch - '0';
19         }
20         return ret;
21     }
22 } buf;

```

7.3 splay

```
1 struct node
2 {
3     static const int INF = 100000000;
4     node* ch[2], *pre;
5     int v, flip, minn, delta, tot;
6     node(int v, int tot, node* l, node* r, node* pre)
7         : v(v), minn (v), tot(tot), pre(pre)
8     {
9         this->delta = 0;
10        this->flip = 0;
11        ch[0] = l, ch[1] = r;
12    }
13    inline int minValue()
14    {
15        return minn;
16    }
17    inline int size()
18    {
19        return tot;
20    }
21    void reverse()
22    {
23        if(tot == 0) return;
24        flip ^= 1;
25    }
26    void add(int d)
27    {
28        if(tot == 0) return;
29        this->minn += d;
30        this->delta += d;
31        this->v += d;
32    }
33    void pushDown()
34    {
35        if(tot == 0) return;
36        if(delta)
37        {
38            if(ch[0]->tot) ch[0]->add(delta);
39            if(ch[1]->tot) ch[1]->add(delta);
40        }
41        if(flip)
42        {
43            swap(ch[0], ch[1]);
44            if(ch[0]->tot) ch[0]->reverse();
45            if(ch[1]->tot) ch[1]->reverse();
46        }
47        flip = delta = 0;
48    }
49    void pushUp()
50    {
51        if(tot == 0) return;
52        tot = ch[0]->size() + ch[1]->size() + 1;
53        minn = min(v, min(ch[0]->minValue(), ch[1]->minValue()));
54    }
55 };
56 class splayTree
57 {
58 private:
59     node* root, * null;
60     void clear(node*& now)
61     {
62         if(now == null) return;
63         clear(now->ch[0]);
```

```

64     clear(now->ch[1]);
65     delete now;
66     now = null;
67 }
68 void rotate(node* x, int type)
69 {
70     node *y = x->pre;
71     y->pushDown();
72     x->pushDown();
73     y->ch[!type] = x->ch[type];
74     if (x->ch[type] != null) x->ch[type]->pre = y;
75     x->pre = y->pre;
76     if (y->pre != null)
77     {
78         if (y->pre->ch[0] == y) y->pre->ch[0] = x;
79         else y->pre->ch[1] = x;
80     }
81     x->ch[type] = y, y->pre = x;
82     if (y == root) root = x; // root 表示整棵树的根结点
83     y->pushUp();
84     x->pushUp();
85 }
86 void splay(node* x, node* f)
87 {
88     x->pushDown();
89     while(x->pre != f)
90     {
91         if (x->pre->pre == f)
92         {
93             if (x->pre->ch[0] == x) rotate(x, 1);
94             else rotate(x, 0);
95         }
96         else
97         {
98             node *y = x->pre;
99             node *z = y->pre;
100             if (z->ch[0] == y)
101             {
102                 if (y->ch[0] == x) // 一字形旋转
103                     rotate(y, 1), rotate(x, 1);
104                 else // 之字形旋转
105                     rotate(x, 0), rotate(x, 1);
106             }
107             else
108             {
109                 if (y->ch[1] == x) // 一字形旋转
110                     rotate(y, 0), rotate(x, 0);
111                 else // 之字形旋转
112                     rotate(x, 1), rotate(x, 0);
113             }
114         }
115     }
116     x->pushUp();
117 }
118 void build(int l, int r, node*& now, node* pre, int* val)
119 {
120     if(l > r) return;
121     int mid = (l + r) / 2;
122     now = new node(val[mid], 1, null, null, pre);
123     build(l, mid - 1, now->ch[0], now, val);
124     build(mid + 1, r, now->ch[1], now, val);
125     now->pushUp();
126 }
127 // the flag node is !not! included, be careful when make interval
128 void findK(int k, node* pre)
129 {

```

```

130     node* now = root;
131     while(true)
132     {
133         now->pushDown();
134         int s = now->ch[0]->size();
135         if(s == k)
136             break;
137         else if(s > k)
138             now = now->ch[0];
139         else
140         {
141             now = now->ch[1];
142             k -= s + 1;
143         }
144     }
145     splay(now, pre);
146 }
147 void makeInterval(int a, int b)
148 {
149     findK(a - 1, null);
150     findK(b + 1, root);
151 }
152 public:
153     splayTree()
154     {
155         null = new node(node::INF, 0, 0, 0, 0);
156         root = null;
157     }
158     ~splayTree()
159     {
160         clear(root);
161         delete null;
162     }
163     // make a sequence from 1 to n do build(0, n + 1, val)
164     // and make sure val[0] = va[1] = INF;
165     void build(int l, int r, int* val)
166     {
167         if(l > r) return;
168         build(l, r, root, null, val);
169     }
170 #define centre (root->ch[1]->ch[0])
171     int minElement(int a, int b)
172     {
173         makeInterval(a, b);
174         return centre->minValue();
175     }
176     void addValue(int a, int b, int value)
177     {
178         makeInterval(a, b);
179         centre->add(value);
180         splay(centre, null);
181     }
182     void reverse(int a, int b)
183     {
184         if(a == b) return;
185         makeInterval(a, b);
186         centre->reverse();
187         splay(centre, null);
188     }
189     void revolve(int a, int b, int c)
190     { // c < b - a + 1, revolve right
191         if(c == 0) return;
192         int len = b - a + 1;
193         reverse(a, a + len - c - 1);
194         reverse(a + len - c, b);
195         reverse(a, b);

```

```

196     }
197     void insert(int a, int c)
198     {
199         makeInterval(a + 1, a);
200         centre = new node(c, 1, null, null, root->ch[1]);
201         root->ch[1]->pushUp();
202         root->pushUp();
203         splay(centre, null);
204     }
205     void erase(int a)
206     {
207         makeInterval(a, a);
208         delete centre;
209         centre = null;
210         root->ch[1]->pushUp();
211         root->ch[0]->pushUp();
212     }
213     void clear()
214     {
215         clear(root);
216     }
217 } tree;
218 const int N = 300000;
219 int val[N];
220
221 int main()
222 {
223     int n, m;
224     int a, b, c;
225     char cmd[100];
226     while(scanf("%d", &n) == 1)
227     {
228         for(int i = 1; i <= n; i++)
229             scanf("%d", &val[i]);
230         val[0] = val[n + 1] = node::INF;
231         tree.clear();
232         tree.build(0, n + 1, val);
233         scanf("%d", &m);
234         REP(i, 0, m)
235         {
236             scanf("%s", cmd);
237             if(!strcmp(cmd, "ADD"))
238             {
239                 scanf("%d %d %d", &a, &b, &c);
240                 tree.addValue(a, b, c);
241             }
242             else if(!strcmp(cmd, "REVERSE"))
243             {
244                 scanf("%d %d", &a, &b);
245                 tree.reverse(a, b);
246             }
247             else if(!strcmp(cmd, "REVOLVE"))
248             {
249                 scanf("%d %d %d", &a, &b, &c);
250                 int tot = b - a + 1;
251                 c = (c % tot + tot) % tot;
252                 tree.revolve(a, b, c);
253             }
254             else if(!strcmp(cmd, "INSERT"))
255             {
256                 scanf("%d %d", &a, &c);
257                 tree.insert(a, c);
258             }
259             else if(!strcmp(cmd, "DELETE"))
260             {
261                 scanf("%d", &a);

```

```

262         tree.erase(a);
263     }
264     else if(!strcmp(cmd, "MIN"))
265     {
266         scanf("%d %d", &a, &b);
267         printf("%d\n", tree.minElement(a, b));
268     }
269 }
270 }
271 return 0;
272 }

```

7.4 network with low bound

```

1  #define OP(i) ((i) - (pool))^1
2  class sap
3  {
4  private:
5      const static int V = 2010, E = 100000, INF = 1000000000;
6      int dis[V], numdis[V], pre[V], b[V];
7      struct edge
8      {
9          int v, cap, low;
10         edge *nxt;
11     } pool[E], *g[V], *pp, *e[V], *pree[V];
12     void bfs(int v, int n)
13     {
14         int que[V], tail = 0;
15         bool vst[V] = {0};
16         memset(numdis, 0, sizeof(numdis));
17         fill(dis, dis + n, n);
18         dis[v] = 0, vst[v] = 1, que[0] = v;
19         for (int j = 0; j <= tail; j++)
20         {
21             int tmp = que[j % n];
22             for (edge *i = g[tmp]; i != NULL; i = i->nxt)
23             {
24                 if (pool[OP(i)].cap > 0 && !vst[i->v])
25                 {
26                     tail++;
27                     vst[i->v] = 1;
28                     que[tail % n] = i->v;
29                     dis[i->v] = dis[tmp] + 1;
30                     numdis[dis[i->v]]++;
31                 }
32             }
33         }
34     }
35     int findArgumentPath(int &v, int s, int t)
36     {
37         while (e[v] != NULL)
38         {
39             if (e[v]->cap > 0 && dis[v] == dis[e[v]->v] + 1)
40             {
41                 pre[e[v]->v] = v, pree[e[v]->v] = e[v], v = e[v]->v;
42                 if (v == t)
43                 {
44                     int minf = INF;
45                     for (int i = t; i != s; i = pre[i])
46                         minf = min(minf, pree[i]->cap);
47                     for (int i = t; i != s; i = pre[i])
48                     {
49                         pree[i]->cap -= minf;
50                         pool[OP(pree[i])].cap += minf;

```

```

51         }
52         v = s;
53         return minf;
54     }
55     }
56     else e[v] = e[v]->nxt;
57 }
58 return 0;
59 }
60 void createEdge(int i, int j, int cap, int low)
61 {
62     pp->v = j, pp->low = low;
63     pp->cap = cap, pp->nxt = g[i];
64     g[i] = pp++;
65 }
66 public:
67 void addedge(int i, int j, int cap, int low)
68 {
69     createEdge(i, j, cap - low, low);
70     createEdge(j, i, 0, low);
71     b[j] += low, b[i] -= low;
72 }
73 int getLimit(int n, int s, int t)
74 {
75     int tmpans = 0;
76     for(int i = 0; i < n; i++)
77     {
78         if(i == s || i == t) continue;
79         if(b[i] > 0) tmpans += b[i];
80     }
81     return tmpans;
82 }
83 int maxflowsap(int n, int s, int t)
84 { //n points is from 1 to n, src is 0, tar is n + 1
85     for(int i = 0; i < n; i++)
86     {
87         if(i == s || i == t) continue;
88         if(b[i] < 0) addedge(i, t, -b[i], 0);
89         if(b[i] > 0) addedge(s, i, b[i], 0);
90     }
91     bfs(t, n);
92     int v = s, maxflow = 0;
93     copy(g, g + n, e);
94     while (dis[s] < n)
95     {
96         int add = findArgumentPath(v, s, t);
97         maxflow += add;
98         if (add == 0)
99         {
100             int mindis = n;
101             numdis[dis[v]]--;
102             if (!numdis[dis[v]]) break;
103             for (edge *i = g[v]; i != NULL; i = i->nxt)
104                 if (i->cap > 0) mindis = min(mindis, dis[i->v] + 1);
105             dis[v] = mindis;
106             numdis[dis[v]]++;
107             e[v] = g[v];
108             if (v != s) v = pre[v];
109         }
110     }
111     return maxflow;
112 }
113 void firststart()
114 {
115     pp = pool;
116     memset(g, 0, sizeof(g));

```

```

117     memset(b, 0, sizeof(b));
118 }
119 void getAnswer(int n, int m, int k)
120 {
121     vector<int> ans[V];
122     edge * j;
123     int i = 0;
124     for(i = 0, j = &pool[1]; i < m; i++, j += 2)
125     {
126         if(j->cap == 1)
127             ans[pool[OP(j)].v - n].push_back(j->v);
128     }
129     for(int i = 1; i <= k; i++)
130     {
131         printf("%d", ans[i].size());
132         FOREACH(ans[i], j)
133             printf(" %d", *j);
134         puts("");
135     }
136 }
137 }flow;
138
139 int main()
140 {
141     //freopen("data.in", "r", stdin);
142     int n, k;
143     while(scanf("%d %d", &n, &k) == 2)
144     {
145         int s = 0, t = n + k + 1;
146         int ss = t + 1, tt = t + 2;
147         int tot = 0;
148         flow.firststart();
149         for(int i = 1; i <= n; i++)
150         {
151             int a, c;
152             scanf("%d", &c);
153             tot += c;
154             while(c--)
155             {
156                 scanf("%d", &a);
157                 flow.addedge(i, a + n, 1, 0);
158             }
159         }
160         for(int i = 1; i <= n; i++)
161             flow.addedge(s, i, 1, 0);
162         for(int i = 1; i <= k; i++)
163             flow.addedge(i + n, t, n, 2);
164         flow.addedge(t, s, n, k * 2);
165
166         int tmpans = flow.getLimit(tt + 1, ss, tt);
167         int ff = flow.maxflowsap(tt + 1, ss, tt);
168         if(tmpans == ff)
169         {
170             printf("YES\n");
171             flow.getAnswer(n, tot, k);
172         }
173         else
174             printf("NO\n");
175     }
176     return 0;
177 }

```
