/ICPC 算法册

GDCPC2010 Edition

Foshan University

Zagfai

2010/4/26

目录

目录 1

基本数据结构和处理方法 3

1. 栈的基本实现 3
2. 游标式队列 4
3. 循环队列 5
4. 二叉树结构与遍历 6
5. 图遍历和搜索 (DFS,BFS) 7
6. TRIE 8
7. Hash Table 9
8. Treap (Tree + Heap Static) 9
9. 二叉堆 12
10. 左偏树 (logN 树) 13
11. 并查集 (路径压缩) 14
12. 线段树 (静态) 15
13. 树状数组 16
14. 二维树状数组 17
15. LCA 离线算法 17

字符串问题 19

16. 常用字符串 Hash 函数 19
17. KMP 20
18. Karp-Rabin 20
19. 二维 Karp-Rabin* 21
20. Sunday 匹配 22
21. 后缀数组 O(n*log n)* 23
22. 最短公共祖先* 23
23. 最长公共子串* 23

数的排序与查找 24

24. 低效插入排序 24
25. 随机化快速排序 24
26. 计数排序 24
27. 二分查找 25
28. 二分查找（大于等于 v 的第一个值） 25
29. 查找第 K 小元素 (基于快速排序) 26
30. 逆序对算法 (基于并归排序) 26
31. 逆序对算法 (基于树状数组) 27

数论及基本数学问题 28

32. gcd 28

33. RMQ 28
附录 29
34. 输入 29
35. 取值范围及常用函数 31
36. 位运算 32
37. 优先级表 33
38. 常用数学定理公式 33
39. ASCII 码表 36
参考文献 37

基本数据结构和处理方法

1. 栈的基本实现

```
#define MAX 100
typedef struct
{
    int data[m];
    int top;
} stackdata;
int init(stackdata *s)   /*初始化栈*/
{
    s->top = 0;
    return 1;
}
int isempty(stackdata *s)
{
    if( s->top == 0) return 1;
    return 0;
}
int push(stackdata *s, int x)   /*入栈操作*/
{
    if (s->top == MAX) return 0;
    else
    {
        s->top = s->top+1;
        s->stack[s->top] = x;
        return 1;
    }
}
int pop(stackdata *s)    /*出栈操作并返回被删除的那个记录*/
{
    int y;
    if( isempty(s))
        printf("the stack is empty!\n");
    else
     {
        y = s->stack[s->top];
        s->top = s->top-1;
        return y;
     }
}
```

2. 游标式队列

```
#define maxsize 100
typedef struct{
   int data[maxsize];
   int front;
   int rear;
} queue;
int sqinit(queue *p) /*初始化*/
{
   p->front = 0;
   p->rear = 0;
   return 1;
}
int enqueue(queue *q, int e)     /*入队*/
{
   if((q->rear+1)%maxsize ==q->front)
      return 0;
   else
      q->data[q->rear] = e;
   q->rear = (q->rear+1) %maxsize;
   return 1;
}
int dequeue(queue *q)     /*出队*/
{
   int e;
   if (q->front == q->rear)
      return 0;
   e = q->data[q->front];
   q->front = (q->front+1)%maxsize;
   return e;
}
int isempty(queue *q)   /*判空*/
{
   int v;
   if (q->front==q->rear)
      v = 1;
   else
      v = 0;
    return v;
}
```

3. 循环队列

```
#define MAX 100
typedef struct
{
	int h;
	int t;
	int data[MAX];
}cycque;

void init(cycque *q)
{
	q->h = q->t = 0;
}
int isempty(cycque *q)
{
	return q->h == q->t;
}
int isfull(cycque *q)
{
	return (( (q->t+1)%MAX == q->h)?1:0);
}
int q_out(cycque *q)
{
	if(isempty(q)) return -32767;
	return q->data[ (q->h+1)%MAX ];
}
int q_in(cycque *q, int x)
{
	if(isfull(q)) return 0;
	q->t = (q->t + 1)%MAX;
	q->data[ q->t ] = x;
	return 1;
}
int q_del(cycque *q)
{
	if(isempty(q)) return -32767;
	q->h = (q->h+1)%MAX;
	return q->data[ q->h ];
}
```

4. 二叉树结构与遍历

```
typedef struct bnode
{
	int data;
	struct bnode *lchild, *rchild;
}node, *root;
int maketree(node *t)
{
	if( t = malloc(sizeof(node)))
	{
		data = 0;
		t->lchild = t->rchild = 0;
		return 1;
	}
	return 0;
}
void preorder(node *t)
{
	if(t)
	{
		printf("%d", t->data);
		preorder( t->lchild );
		preorder( t->rchild );
	}
	return;
}
void inorder(node *t);
void postorder(node *t);
void levelorder(node *t)
{
	queue q;
	init(q);
	enqueue(q, t);
	while(!isempty(q))
	{
		node *T = dequeue(q);
		printf("%d", T->data);
		if( T->lchild ) enqueue(q, T->lchild);
		if( T->rchild ) enqueue(q, T->rchild);
	}
}
```

5. 图遍历和搜索 (DFS,BFS)

```
/*多次 dfs 或 bfs 即对图进行遍历*/
/*  以下为邻接矩阵实现*/
char mark[100];
char g[100][100];
int n;   /*节点数*/
void dfs(int i)
{
      int j;
      mark[i] = 1;
      for(j=0; j<n ;j++)
            if(g[i][j] && !mark[j])
            {
                  mark[j] = 1;
                  dfs(j);
            }
      return ;
}
void bfs(int i)
{
      int j, k;
      mark[i] = 1;
      queue q;
      init(q);
      enqueue(q, i);
      while(!isempty(q))
      {
            j = dequeue(q);
            for(k=0; k<n ;k++)
                  if(g[j][k] && !mark[k])
                  {
                        printf("%d", j);
                        mark[k] = 1;
                        enqueue(q, k);
                  }
      }
}
```

6. TRIE

```
#define pos(x) (x-97)
struct poi
{
        struct poi *next[27];
        char w;
        int key;
}root;
void format(struct poi *a)      /*格式化节点*/
{
      int i;
      for(i=0; i<27 ;i++) a->next[i] = NULL;
}
void in(char *word, int key)   /*插入*/
{
      int i=-1;
      struct poi *p = (struct poi *) &root;
      while(word[++i] !=0)
      {
                  if(p->next[ pos(word[i]) ] == NULL)
                  {
                        p->next[ pos(word[i])] = malloc(sizeof(struct poi));
                        p = p->next[ pos(word[i])];
                        format(p);
                        p->w=word[i];
                        if(word[i+1] ==0) p->key = key;
                  }
                  else    p = p->next[ pos(word[i]) ];
      }
}
int find(char *word)  /*查找*/
{
     struct poi *p = (struct poi *) &root;
     int i;
     for(i=0; word[i] && p ;i++)
              p=p->next[ pos(word[i]) ];
     return word[i]?-1:p->key;
}
```

7. HASH TABLE

```
/*997    9973    29989   49999*/
#define TSIZE 1000
typedef struct hash_table
{
    int data;
    int *next;
}HT;
HT ht[TSIZE];
int h(int x)
{
    return x%997;
}
void into_table(int data)
{
    HT *p = &(ht[h(data)]);
    while(p->next != NULL)
    {
        if(p->data == data) return;
        p = p->next;
    }
    p->next = malloc(sizeof(HT));
    p = p->next;
    p->next = NULL;
    p->data = data;
    return;
}
int find(int data)
{
    HT *p = &(ht[h(data)]);
    while(p->next != NULL)
    {
        if(p->data == data) return 1;
        p = p->next;
    }
    return 0;
}
```

8. TREAP (TREE + HEAP STATIC)

```
struct poi{
        long aux, key, l, r, num;
}s[100000];
```

```
long point_count, root;
void lrotate(long *p)
{
	long p1 = s[*p].l;
	s[*p].l = s[p1].r;
	s[p1].r = *p;
	*p = p1;
}
void rrotate(long *p)
{
	long p1 = s[*p].r;
	s[*p].r = s[p1].l;
	s[p1].l = *p;
	*p = p1;
}
void insert(long *p, long u)
{
	if(0 == *p){
		*p = ++point_count;
		s[*p].key = u;
		s[*p].num = 1;
		s[*p].aux = rand()+1;
		s[*p].l = s[*p].r = 0;
		}
	else if(u < s[*p].key) {
			insert(&(s[*p].l),u);
			if(s[s[*p].l].aux > s[*p].aux) lrotate(p);
			}
	else if(u > s[*p].key) {
			insert(&(s[*p].r),u);
			if(s[s[*p].r].aux > s[*p].aux) rrotate(p);
			}
	else s[*p].num++;
}
void del(long *p, long x)
{
	if	(*p==0)		return;
	else if(s[*p].key <x) del( &(s[*p].r) ,x);
	else if(s[*p].key >x) del( &(s[*p].l) ,x);
	else
	{
		if(s[*p].num>1) s[*p].num--;
		else if(s[*p].l == 0 && s[*p].r == 0) *p = 0;
		else if(s[*p].l == 0 || s[s[*p].l].aux < s[s[*p].r].aux)
```

```
            {
                        rrotate(p);
                        del(p, x);
            }
            else if(s[*p].r == 0 || s[s[*p].l].aux > s[s[*p].r].aux)
            {
                        lrotate(p);
                        del(p, x);
            }
      }
}
int find(long p, long x)
{
      if(p == 0) return 0x7fffffff;
      if       (s[p].key == x) return x;
      else if(s[p].key <   x) return find(s[p].r, x);
      else if(s[p].key >   x) return find(s[p].l, x);
}

void order(long q)
{
       if(q==0) return;
       order(s[q].l);
       printf("%d\n", s[q].key);
       order(s[q].r);
}
int main()
{
      memset(s,0,sizeof(s));
      root = point_count = 1;
      int n = 9;
      int i, j;
      srand(time(NULL));

      for(i=0; i<= n ;i++) insert(&root, i);
      order(root);
      printf("\n");printf("\n");printf("\n");
      for(i=0; scanf("%d", &j) ;i++)
      {
                  printf("%d\n", find(root, j));
                  del(&root, j);
      }
}
```

9. 二叉堆

```
int h[100000];
int size=0;
int pop_heap()
{
	return h[0];
}
void insert_heap(int dat)
{
	int i, t;
	i=size++;
	h[i]=dat;
	while(i>0 && h[i]<h[i/2])
	{
		t = h[i];
		h[i] = h[i/2];
		h[i/2] = t;
		i /= 2;
	}
}
int del_heap()
{
	int l, r, i, m, t;
	size--;
	h[0] = h[size];
	i=0;
	l=1;
	r=2;
	while(1)
	{
		if(l<size && h[l]<h[i]) m = l;
		else m = i;
		if(r<size && h[r]<h[m]) m = r;
		if(m == i) return;
		t = h[i];
		h[i] = h[m];
		h[m] = t;
		i = m;
		l = i+i;
		r = l+1;
	}
}
```

10. 左偏树 (LOGN 树)

```
#define typec int                    // type of key value
const int na = -1;
struct node
{
    typec key;
    int l, r, f, dist;
}tr[N];
int iroot(int i){                    // find i's root
  if (i == na) return i;
    while (tr[i].f != na) i = tr[i].f;
    return i;
}
int merge(int rx, int ry){       // two root: rx, ry
    if (rx == na) return ry;
    if (ry == na) return rx;
    if (tr[rx].key > tr[ry].key) swap(rx, ry);
    int r = merge(tr[rx].r, ry);
    tr[rx].r = r; tr[r].f = rx;
    if (tr[r].dist > tr[tr[rx].l].dist)
        swap(tr[rx].l, tr[rx].r);
    if (tr[rx].r == na) tr[rx].dist = 0;
    else tr[rx].dist = tr[tr[rx].r].dist + 1;
    return rx;                       // return new root
}
int ins(int i, typec key,   int root){   // add a new node(i, key)
    tr[i].key = key;
    tr[i].l = tr[i].r = tr[i].f = na;
    tr[i].dist = 0;
    return root = merge(root, i);   // return new root
}
int del(int i) {                 // delete node i
if (i == na) return i;
    int x, y, l, r;
    l = tr[i].l; r = tr[i].r; y = tr[i].f;
    tr[i].l = tr[i].r = tr[i].f = na;
    tr[x = merge(l, r)].f = y;
    if (y != na && tr[y].l == i) tr[y].l = x;
    if (y != na && tr[y].r == i) tr[y].r = x;
    for ( ; y != na; x = y, y = tr[y].f) {
        if (tr[tr[y].l].dist < tr[tr[y].r].dist)
            swap(tr[y].l, tr[y].r);
        if (tr[tr[y].r].dist + 1 == tr[y].dist) break;
```

```
            tr[y].dist = tr[tr[y].r].dist + 1;
        }
        if (x != na) return iroot(x); // return new root
        else return iroot(y);
        }
node top(int root){
        return tr[root];
}
node pop(int &root){
        node out = tr[root];
        int l = tr[root].l, r = tr[root].r;
        tr[root].l = tr[root].r = tr[root].f = na;
        tr[l].f = tr[r].f = na;
        root = merge(l, r);
        return out;
}
int add(int i, typec val)    // tr[i].key += val
{
    if (i == na) return i;
    if (tr[i].l == na && tr[i].r == na && tr[i].f == na) {
    tr[i].key += val;
      return i;
  }
    typec key = tr[i].key + val;
    int rt = del(i);
    return ins(i, key, rt);
}
void init(int n){
    for (int i = 1; i <= n; i++) {
    scanf("%d", &tr[i].key); //%d: type of key
        tr[i].l = tr[i].r = tr[i].f = na;
    tr[i].dist = 0;
  }
}
```

11. 并查集 (路径压缩)

```
int n, m, p;
int pi[5005];
int father(int who)
{
        if(pi[who] == who) return who;
        else
        {
```

```
            pi[who] = father(pi[who]);
            return pi[who];
        }
}
int find(int a, int b)
{
        int af = father(a);
        int bf = father(b);
        if(af == bf) return 1;
        else          return 0;
}
void insert(int a, int b)
{
        int af = father(a);
        int bf = father(b);
        pi[af] = bf;
}
```

12. 线段树 (静态)

```
//线段树，MAX 表示最大的范围。right, left 表示儿子被覆盖的长度。
//count 表示此段被覆盖的次数。area 是此段被覆盖的长度。
//start 和 end 是此节点代表的起末位置。mid 是中间分割点。
//可以相应的修改一下参数，实现其他功能。
//矩形面积并,周长并在计算几何中
#define MAX 1024*2
struct node
{
    int right, left, count;
    int start, end, mid, area;
}tree[MAX];
void construct(int s, int e, int d) //构建树
{
    tree[d].start = s;
    tree[d].end    = e;
    tree[d].mid    =( s + e )/2;
    if( s+1 == e )   return;
    construct(s, (s+e)/2, d*2+1);
    construct( (s+e)/2, e, d*2+2);
}
void init(){
  int i;
  for(i=0; i<MAX ;i++)
    tree[i].right = tree[i].left = tree[i].count = tree[i].area = 0;
```

```
}
int insert(int s, int e, int d)     //加线段
{
    if( tree[d].start >= s && tree[d].end <= e)     tree[d].count++;
    else
    {
        if( tree[d].mid > s ) tree[d].left = insert(s, e, d*2+1);
        if( tree[d].mid < e ) tree[d].right = insert(s, e, d*2+2);
    }
    if( tree[d].count )
        tree[d].area = tree[d].end-tree[d].start;
    else
        tree[d].area = tree[d].left + tree[d].right;
    return tree[d].area;
}
int del(int s, int e, int d)   //  删线段
{
    if( tree[d].start >=s && tree[d].end <=e) tree[d].count--;
    else
    {
        if( tree[d].mid > s ) tree[d].left = del(s, e, d*2+1);
        if( tree[d].mid < e ) tree[d].right = del(s, e, d*2+2);
    }
    if( tree[d].count )
        tree[d].area = tree[d].end - tree[d].start;
    else
        tree[d].area = tree[d].left + tree[d].right;
    return tree[d].area;
}
```

13. 树状数组

```
int c[32003];
int max=32003;
void insert(int k, int delta)
{
    for(; k<=max ;k+=(-k)&k ) c[k] +=delta;
}
int getsum(int k) /* 0~k 和*/
{
    int t;
    for(t=0; k>0 ;k-=(-k)&k ) t +=c[k];
    return t;
}
```

14. 二维树状数组

```
int N = 10000;
int c[10000][10000];
int Row, Col;
int Lowbit(const int &x)
{
    return x&(-x);
}
int getsum(int i, int j)
{
    int  tempj, sum = 0;
    while( i > 0 )
    {
        tempj = j;
        while( tempj > 0 )
        {
        sum += c[i][tempj];
        tempj -= Lowbit(tempj);
        }
        i -= Lowbit(i);
    }
    return sum;
}
void modify(int i, int j, int delta)    /* (0, 0) to (i, j) 变化 delta*/
{
    int  tempj;
    while( i <= Row )
    {
        tempj = j;
        while( tempj <= Col )
        {
            c[i][tempj] += delta;
            tempj += Lowbit(tempj);
        }
        i += Lowbit(i);
    }
}
```

15. LCA 离线算法

```
/*  O(E)+O(1) 使用邻接表可优化为 O(E)+O(1)
    INIT: id[]置为-1; g[]置为邻接矩阵
    结果为 lcs[][]
```

```
*/
#define N 100
int id[N], lcs[N][N], g[N][N];
int getpa(int i)
{
	if(id[i] == i) return i;
	return id[i] = getpa(id[i]);
}
void dfs(int rt, int n)
{
	int i;
	id[rt] = rt;
	for(i = 0; i < n; ++i)
		if(g[rt][i] && -1 == id[i])
		{
			dfs(i, n);
			id[get(i)] = getpa(rt);
		}
	for(i = 0; i < n; ++i)
		if (-1 != id[i])
			lcs[rt][i] = lcs[i][rt] = getpa(i);
}
void LCA(int n)
{
	int i;
	for (i=0; i<n; ++i)
			if (id[i] == -1) dfs(i, n);
}
```

字符串问题

16. 常用字符串 HASH 函数

```
unsigned int hasha(char *url, int mod)
{
	unsigned int n = 0;
	int i;
	char *b = (char *) &n;
	for(i=0; url[i] ; ++i)
		b[i % 4] ^= url[i];
	return n % mod;
}
unsigned int hashb(char *url, int mod)
{
	unsigned int h = 0, g;
	while (*url)
	{
		h = (h << 4) + *url++;
		g = h & 0xF0000000;
		if(g)
			h ^= g >> 24;
		h &= ~g;
	}
	return h % mod;
}
int hashc(char *p, int prime = 25013)
{
	unsigned int h=0, g;
	for(; *p ;++p)
	{
		h = (h<<4) + *p;
		if(g = h & 0xf0000000)
		{
			h = h ^ (g >> 24);
			h = h ^ g;
		}
	}
	return h % prime;
}
```

17. KMP

```
// 模式匹配 ,kmp 算法 , 复杂度 O(m+n)
// 返回匹配位置 ,-1 表示匹配失败 , 传入匹配串和模式串和长度
// 可更改元素类型 , 更换匹配函数
#define MAXN 10000
#define _match(a, b) ((a)==(b))
int fail[MAXN];
int pat_match(char *str, char *pat, int ls, int lp)
{
    int i, j;
    for(i=0; i < MAXN ;i++) fail[i] = -1;

    for(i=0,j=1; j < lp ;j++)
    {
        for(i=fail[ j-1]; i>=0 && !_match(pat[i+1],pat[j]); i=fail[i]) ;
        fail[j] = (_match(pat[i+1],pat[j]) ? i+1 : -1);
    }
    for(i=j=0; i<ls && j<lp ;i++)
        if(_match(str[i],pat[j]))
            j++;
        else if (j)
            j=fail[ j-1]+1,i--;
    return j==lp ? (i-lp):-1;
}
```

18. KARP-RABIN

```
#define REHASH(a, b, h) ((((h) - (a)*d) << 1) + (b))
int KR(char *text, char *patent, int text_len, int patent_len)
{
    int d, hx, hy, i, j;
    for(d=i=1; i < patent_len ;i++) d <<= 1;
    for(hy=hx=i=0; i < patent_len ;i++)
    {
        hx = ((hx<<1) + text[i]);
        hy = ((hy<<1) + patent[i]);
    }
    for(j=0; j <= text_len - patent_len ;j++)
    {
        if( hx==hy && memcmp(text, patent + j, patent_len)==0 ) return j;
        hy = REHASH(patent[j], patent[j + patent_len], hy);
    }
}
```

19. 二维 KARP-RABIN*

```
#define uint unsigned int
const int A=1024, B=128;
const uint E=27;
char text[A][A], patt[B][B];
uint ht, hp, pw[B * B], hor[A], ver[A][A];
int n, m, x, y;
void init(){
   int i, j = B * B;
   for (i=1, pw[0]=1; i<j; ++i) pw[i] = pw[i-1] * E;
}
void hash(){
   int i, j;
   for (i=0; i<n; ++i) for (j=0, hor[i]=0; j<y; ++j) {
   hor[i]*=pw[x]; hor[i]+=text[i][j]-'a';
 }
   for (j=0; j<m; ++j) {
     for (i=0, ver[0][j]=0; i<x; ++i) {
     ver[0][j]*=E; ver[0][j]+=text[i][j]-'a';
   }
     for (i=1 ; i<=n-x; ++i)
     ver[i][j]=
     (ver[i-1][j]-(text[i-1][j]-'a')*pw[x-1])*E
     +text[i+x-1][j]-'a';
 }
   for (j=0, ht=hp=0; j<y; ++j) for (i=0; i<x; ++i) {
   ht*=E; ht+=text[i][j]-'a';
   hp*=E; hp+=patt[i][j]-'a';
 }
}
void read(){
   int i;
 scanf("%d%d", &n, &m);
   for (i=0; i<n; ++i) scanf("%s", text[i]);
 scanf("%d%d", &x, &y);
   for (i=0; i<x; ++i) scanf("%s", patt[i]);
}
int solve(){
   if (n==0||m==0||x==0||y==0) return 0;
   int i, j, cnt=0; uint t;
   for (i=0; i<=n-x; ++i) {
     for (j=0, t=ht; j<=m-y; ++j) {
       if (t==hp) ++cnt;
```

```
  t=(t-ver[i][j]*pw[y*x-x])*pw[x]+ver[i][j+y];
   }
   ht=(ht-hor[i]*pw[x-1])*E+hor[i+x];
 }
   return cnt;
}
int main(void){
   int T; init();
   for (scanf("%d", &T); T; --T) {
   read(); hash();
   printf("%d\n", solve());
 }
   return 0;
}
```

20. SUNDAY 匹配

```
int sunday(char *src, char *des)
{
	int i, j, pos=0;
	int len_s, len_d;
	int next[26];

	len_s = strlen(src);
	len_d = strlen(des);
	for(j=0; j<26 ;j++) next[j]=len_d;
	for(j=0; j<len_d ;++j)
		next[des[j]-'a'] = len_d-j;
	while( pos < (len_s-len_d+1) )
	{
		i = pos;
		for(j=0; j<len_d ;j++,i++)
		{
			if(src[i] != des[j])
			{
				pos += next[ src[pos + len_d] - 'a' ];
				break;
			}
		}
		if(j == len_d) return pos;
	}
	return -1;
}
```

21. 后缀数组 O(N*LOG N)*

```
char s[N];                                  // N > 256
int n, sa[N], height[N], rank[N], tmp[N], top[N];
void makesa(){                              // O(N * log N)
  int i, j, len, na;
  na = (n < 256 ? 256 : n);
  memset(top, 0, na * sizeof(int));
  for (i = 0; i < n ; i++) top[ rank[i] = s[i] & 0xff ]++;
  for (i = 1; i < na; i++) top[i] += top[i - 1];
  for (i = 0; i < n ; i++) sa[ --top[ rank[i] ] ] = i;
  for (len = 1; len < n; len <<= 1) {
   for (i = 0; i < n; i++) {
       j = sa[i] - len; if (j < 0) j += n;
       tmp[ top[ rank[j] ]++ ] = j;
  }
     sa[ tmp[ top[0] = 0 ] ] = j = 0;
   for (i = 1; i < n; i++) {
     if (rank[ tmp[i] ] != rank[ tmp[i-1] ] ||
     rank[ tmp[i]+len ]!=rank[ tmp[i-1]+len ])
     top[++j] = i;
        sa[ tmp[i] ] = j;
  }
     memcpy(rank, sa , n * sizeof(int));
     memcpy(sa    , tmp, n * sizeof(int));
   if (j >= n - 1) break;
 }
}
void lcp(){                                 // O(4 * N)
  int i, j, k;
  for (j = rank[height[i=k=0]=0]; i < n - 1; i++, k++)
   while (k >= 0 && s[i] != s[ sa[j-1] + k ])
       height[j] = (k--), j = rank[ sa[j] + 1 ];
}
```

22. 最短公共祖先*

```

```

23. 最长公共子串*

```

```

数的排序与查找

24. 低效插入排序

```
void esort(int *x, int l, int h)
{
	int i,j,t;
	for(i=l; i<=h ;i++)
		for(j=i; j>l ;j--)
			if(x[j] < x[j-1])
			{
				t = x[j];
				x[j] = x[j-1];
				x[j-1] = t;
			}
			else break;
}
```

25. 随机化快速排序

```
void qsort(int *x,int l,int r)
{
	int k =x[l+rand()%(r-l)];
	int i=l,j=r;
	int t;
	while (1)
	{
		while (x[i]<k) i++;
		while (k<x[j]) j--;
		if (i>j) break;

		t=x[i];
		x[i]=x[j];
		x[j]=t;
		i++;
		j--;
	}
	if (i<r) qsort(x,i,r);
	if (l<j) qsort(x,l,j);
}
```

26. 计数排序

```
int count[100000];
void csort(int *x, int l, int h)
{
    int i;
    memset(count,0,sizeof(count));
    for(i=l; i<=h ;i++) count[x[i]]++;
    for(i=0; i<=len(count) ;i++)
        while(count[i])
        {
            count[i]--;
            x[l++]= i;
        }
}
```

27. 二分查找

```
/*  输入: 有序 数组 x ,被查找元素 it, 低位 l, 高位 h
    返回: 找到则返回位置，否则返回 -1*/
int binary_search(int *x, int it, int l, int h)
{
    int mid;
    while(l<=h)
    {
        mid = (l+h)>>1;
        if (it==x[mid])      return mid;
        else if (it<x[mid]) h=mid-1;
        else                 l=mid+1;
    }
    return -1;
}
```

28. 二分查找（大于等于 v 的第一个值）

```
int bs_first_biger(int *x, int it, int l, int h)
{
    int m;
    while ( l<h )
    {
        m = ( l+h )>>1;
        if (x[m] < it)   l=m+1;
        else             h=m;
    }
    return l;
}
```

29. 查找第 K 小元素 (基于快速排序)

```
/*   输入: 数组 x, Kth, 低位 l, 高位 h
     返回: 找到则返回位置，否则返回 -1 */
int partition(int *x, int l, int r)
{
    if (l < r)
    {
        int i = l;
        int j = r;
        int k = x[i];
        while (i < j)
        {
            while (i < j && x[j] > k) j--;
            if (i < j) x[i++] = x[j];
            while (i < j && x[i] < k) i++;
            if (i < j) x[j--] = x[i];
        }
        x[i] = k;
        return i;
    }
    return -1;
}
int selectKth(int *x, int k, int l, int r)
{
    if (k <=0 ) return -1;
    if (l > r)   return -1;
    if (l == r) return x[l];
    int i = partition(x, l, r);
    int j = i - l + 1;
    if (j == k)       return x[i];
    else if (j > k) return selectKth(x, k, l, i);
    else                  return selectKth(x, k-j, i+1, r);
}
```

30. 逆序对算法 (基于并归排序)

```
/*
输入: 数组 x, 低位 l,  数 R 个
返回: 逆序对数量
注意: 先定义有用的数组 c
*/
int c[LongOf_X];
int inversion_pair(int *x, int l, int r)
```

```
{
    int cnt, mid, i, j, tmp;
    cnt = 0;
    if( r > l+1 )
    {
        mid = (l+r)>>1;
        cnt+= inversion_pair(x, l, mid);
        cnt+= inversion_pair(x, mid, r);
        tmp = l;
        for(i=l, j=mid; i<mid && j<r ;)
        {
            if( x[i]>x[j] )
            {
                c[tmp++] = x[j++];
                cnt += mid-i;
            }
            else c[tmp++] = x[i++];
        }
        if( j<r )
            for( ; j<r ; ++j ) c[tmp++] = x[j];
        else
            for( ; i<mid; ++i ) c[tmp++] = x[i];
        for (i=l; i<r; ++i )    x[i] = c[i];
    }
    return cnt;
}
```

31. 逆序对算法 (基于树状数组)

```
int main()
{
    int i,k;
    ans=0;
    for(i=1; i<=n ;i++)
    {
            scanf("%d", &k);
            modify(k);
            ans+=i-get(k);
    }
    printf("%d\n", ans);
    return 0;
}
```

数论及基本数学问题

32. GCD

```
```

33. RMQ

```
#define N 100
int id[N], lcs[N][N], g[N][N];
int getpa(int i)
{
    if(id[i] == i) return i;
    return id[i] = getpa(id[i]);
}
void dfs(int rt, int n)
{
    int i;
    id[rt] = rt;
    for(i = 0; i < n; ++i)
        if(g[rt][i] && -1 == id[i])
        {
            dfs(i, n);
            id[get(i)] = getpa(rt);
        }
    for(i = 0; i < n; ++i)
        if (-1 != id[i])
            lcs[rt][i] = lcs[i][rt] = getpa(i);
}
void LCA(int n)
{
    int i;
```

附录

34. 输入

1) 先输入 n，代表有 n 组数据

```
scanf("%d", &n);
f| (i=0; i<n; i++)
{
    // 根据题目进行处理
}
```

2) 以一个整数（例如 0）表示结束

```
while (scanf("%d", &n) && n!=0)
{
        // 根据题目进行处理
}
```

另一种方法：

```
scanf("%d", &n);
while (n != 0)
{
    // 根据题目进行处理
        scanf("%d", &n);
}
```

3) 以两个整数（例如 0 0）表示结束

```
while (1)
{
    scanf("%d%d", &n, &m);
    if (n==0 && m==0) break;
        // 根据题目进行处理
}
```

另一种方法：

```
scanf("%d%d", &n, &m);
while (n!=0 || m!=0)
{
    // 根据题目进行处理
    scanf("%d%d", &n, &m);
}
```

4) 以某种字符串（例如"END"）表示结束

```
char c[101]; // 这个字符数组的长度视题目描述而定，特别要注意给 '\0' 留出存储空间。
```

```
while (1)
{
    gets(c);
    if ( strcmp( c, "END") == 0 ) break;
        // 根据题目进行处理
}
```

另一种写法：

```
gets(c);
while (strcmp(c, "END") != 0)
{
    // 根据题目进行处理
        gets(c);
}
```

5) 以 **EOF** 表示结束

```
while ( scanf("%d", &n) != EOF )
{
    // 根据题目进行处理
}
```

6) **scanf() , printf()**参数

%d , %i	十进制
%o	八进制
%x	十六进制
%u	无符号十进制
%c	字符
%s	字符串
%f	浮点数
%e	指数
%g	智能浮点输出

- ✧ 输出

  %-m.nd

  m 数据域长度

  n 对实数输出小数点后 **n** 位，对串则截取前 **n** 位输出

  - 域内右靠

- ✧ 输入

  “*” 跳过指定数字或字母，即不作输入

- ✧ 其他常用输入输出

  getch() putchar() puts() gets()

35. 取值范围及常用函数

1. 类型

类型	取值范围
float	**6** 位有效数字，**$10^{\pm 38}$**
double	**15** 位有效数字，**$10^{\pm 308}$**
long double	**18** 位有效数字，**$10^{\pm 4932}$**
long long	**$2^{64}-1$**
__int64	**$2^{64}-1$**

2. 函数

数学函数	**<math.h>**
int abs(int x) **double fabs(double x)**	绝对值**|x|**
double floor(double x)	不大于 **x** 的最大整数
double log10(double x)	**$\log_{10}(x)$** 的值
double pow(double X, double y)	**X^y**
double sqrt(double x)	**$\sqrt{x}$**，确保 **$x \geq 0$**
int r&(void)	**1~32767** 随机数，种子设定 **sr&(int n)**

字符串函数	**<string.h>**
char *strcat(char *str1, char *str2)	**str2** 接到 **str1** 后，添加 **'\0'**，返回 **str1** 指针
char *strchr(char *str, int ch)	**str** 中 **ch** 第一次位置，返回 **ch** 地址,否则 **NULL**
int strcmp(char *str1, char *str2)	字典序比较，返回 **str1 – str2**，相等为 **0**
char *strcpy(char *str1, char *str2)	复制 **str2** 到 **str1**，返回 **str1** 地址
unsigned int strlen(char *str)	返回字符串长度
char *strstr(char *str1, char *str2)	**str1** 中 **str2** 第一次位置，返回地址,否则 **NULL**
memmove(*dest, *src, size)	由 **src** 到 **dest** 移动 **size** 大小
memset(*s, int c, size)	
memcpy(*s, int c, size)	

动态分配函数	**<stdlib.h>**
void *malloc(unsigned size)	分配 **size** 字节并返回起始地址
void free(void *p)	释放 **p** 所指向地址的空间

36. 位运算

简单应用（出自 **matrix67.com**）

功能	示例	位运算
去掉最后一位	**(101101->10110)**	**x >> 1**
在最后加一个 **0**	**(101101->1011010)**	**x << 1**
在最后加一个 **1**	**(101101->1011011)**	**x << 1 +1**
把最后一位变成 **1**	**(101100->101101)**	**x | 1**
把最后一位变成 **0**	**(101101->101100)**	**x | 1 -1**
最后一位取反	**(101101->101100)**	**x ^ 1**
把右数第 **k** 位变成 **1**	**(101001->101101,k=3)**	**x | (1 << (k-1))**
把右数第 **k** 位变成 **0**	**(101101->101001,k=3)**	**x & ! (1 << (k-1))**
右数第 **k** 位取反	**(101001->101101,k=3)**	**x ^ (1 << (k-1))**
取末三位	**(1101101->101)**	**x & 7**
取末 **k** 位	**(1101101->1101,k=5)**	**x & (1 << k-1)**
取右数第 **k** 位	**(1101101->1,k=4)**	**x >> (k-1) & 1**
把末 **k** 位变成 **1**	**(101001->101111,k=4)**	**x | (1 << k-1)**
末 **k** 位取反	**(101001->100110,k=4)**	**x ^ (1 << k-1)**
把右边连续的 **1** 变成 **0**	**(100101111->100100000)**	**x & (x+1)**
把右起第一个 **0** 变成 **1**	**(100101111->100111111)**	**x | (x+1)**
把右边连续的 **0** 变成 **1**	**(11011000->11011111)**	**x | (x-1)**
取右边连续的 **1**	**(100101111->1111)**	**(x ^ (x+1)) >> 1**
删右起第一个 **1** 的左边	**(100101000->1000)**	**x & (x ^ (x-1))**

某数二进制中的 **1** 有奇数个还是偶数个

```
int odd_even_1(int x)
{
    x^= x>>1;
    x^= x>>2;
    x^= x>>4;
    x^= x>>8;
    x^= x>>16;
    return x&1;
}
```

计算二进制中的 **1** 的个数**(32** 位**)**

```
int how_many_1(int x)
{
    x = (x & 0x55555555) + ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x & 0x0F0F0F0F) + ((x >> 4) & 0x0F0F0F0F);
    x = (x & 0x00FF00FF) + ((x >> 8) & 0x00FF00FF);
    x = (x & 0x0000FFFF) + ((x >> 16) & 0x0000FFFF);
    return x;
}
```

37. 优先级表

由上到下优先级递减

关联符	**( ) [ ] -> .**
一元符	**! ~ ++ -- +**(正) **- (type) * &**(取地址) **sizeof()** 从右到左
运算符	*** / %** **+ -**
位移	**<< >>**
比较符	**< <= >= > == !=**
位操作	**&** **^** **|**
逻辑符	**&&** **||**
条件符	**(A)?(B):(C)**
赋值	**= += /= ……**(从右到左)
逗号	**,**

38. 常用数学定理公式

1. 同余式性质
   a) **a≡a (mod m)** 自反性
   b) **a≡b (mod m), b≡c (mod m)** 则 **a≡c (mod m)** 传递性
   c) **(a*b)%c == [(a%c)*b]%c**

2. 常见数学模型
   a) 递推公式与通项公式
   b) 等差数列 公差 **d**, 首项 a_1, 前 **n** 项和 S_n
      i. $a_n = a_{n-1} + d$
      ii. $a_n = a_1 + (n-1)*d$
      iii. $S_n = na_1 + (n(n-1)/2)*d$
      iv. $2a_n = a_{n+1} + a_{n-1}$
   c) 等比数列 公比 **q**, 首项 a_1, 前 **n** 项和 S_n
      i. $a_n = a_{n-1} * q$
      ii. $a_n = a_1 * q^{n-1}$
      iii. $S_n = (a_1 – a_nq)/(1-q)$
      iv. $a_n^2 = a_{n-1} * a_{n+1}$
   d) 常见数列和
      i. $a_n = n \Rightarrow S_n = n(n+1)/2$
      ii. $a_n = n^2 \Rightarrow S_n = n(n+1)(2n+1)/2$
      iii. $a_n = n^3 \Rightarrow S_n = n^2(n+1)^2/4$
   e) 常见模型

i. **Fibonacci** **F(n) = F(n-1) + F(n-2)**

F(n)	1	1	2	3	5	8	13	21	34	55
n	1	2	3	4	5	6	7	8	9	10

ii. **Catalan**

C(n) ?	1	1	2	5	14	42	132
n	1	2	3	4	5	6	7	8

C(n) = [1/(n-1)] * C_{2n-4}^{n-2}

常见应用

- ✧ **n** 个节点可以构成不同形态二叉树数
- ✧ 将一凸多边形分割成三角形区域方案数
- ✧ 进栈序列为 **1,2,3,4…n** 的出栈方案数

iii. **Stirling** 数

S(n, 0) = 0

S(n, 1) = 1 (k = 1)

S(n, n) = 1

S(n, k) = 0 (k > n)

S(n, k) = S(n-1,k-1) + k*S(n-1,k) (n >= k >= 2)

iv. 汉诺塔递推式

$a_n = 2a_{n-1} + 1$

$a_n = 2^n - 1$

v. 分平面最大区域数

1. 直线分平面

2, 4, 7, 11 ……

f(n) = f(n-1) +n **f(n) = n(n+1)/2 +1**

2. 封闭曲线分平面

2, 4, 8, 14 ……

f(n) = f(n-1) + 2(n - 1) **$f(n) = n^2 - n + 2$**

3. 折线分平面

2, 7, 16, 29 ……

f(n) = (n-1)(2n-1) +2n

f) 排列组合

i. 排列 **P(m, n) = n! / (n-m)!**

- ➢ 可重复排列 : **n** 个元素抽 **m** 个，每一次抽完立即放回 n^m
- ➢ 循环排列 : **n** 个元素取 **m** 个，有方向排成圆圈

n!/m*(n-m)!

- ➢ 错位排列 : 发 **n** 封信,完全发错邮筒 **n!(1/2! – 1/3! + … 1/n!)**
- ➢ 不尽相异元素全排列 **m** 个元素，n_1 个 a_1 , … n_n 个 a_n

$$\frac{m!}{n_1! * \dots * n_n!}$$

ii. 组合 **C(m, n) = n! / (m! * (n-m)!)**

- ➢ 性质

- **C(m, n) = C(n-m, n)**
- **C(m, n+1) = C(m, n) + C(m-1, n)**

➢ 不同元素重复组合

从 **n** 种无限个不同元素中，每次取 **r** 个，**r** 个元素允许相同

H(r,n) = C(r, n+r-1)

iii. 应用定理

1. 鸽笼原理

a) 基本原理：**n+1** 物体放进 **n** 个盒子，至少有 **1** 个盒子包含两个以上物体。

b) 平均定理：**n** 个非负整数平均数等于 **r**，则至少有一个数大于 **r**，一个数小于 **r**。

2. **Ramsey** 定理

a) 狭义 **Ramsey** 定理：任意六个人中要么至少三个人认识，要么至少三个不认识

b) 设 **p, q** 为正整数，**p, q ≥ 2**，则存在最小正整数 **R(p, q)**，使得当 **n ≥ R(p, q)** 时，用红蓝两色涂色 K_n 的边，则或存在一个蓝色的 K_p，或存在一个红色的 K_q。

c) 性质： **R(a, b) = R(b, a)**

R(a, 2) = R(2, a) = a

R(a, b) ≤ R(a-1,b) + R(a,b-1)

3. 容斥原理 **A∪B∪C = A+B+C - A∩B - B∩C - C∩A + A∩B∩C**

4. 母函数方法

5. 置换群与 **Pólya** 定理 （求本质不同的染色方案数）

设 **G** 是一个置换群，用 **k** 种颜色染其中元素，方案数 **= (∑ $k^{m(f)}$) / |G|**；

即方案数等于颜色种数 **k** 的 **m(f)**次方除以置换总数；

其中 **m(f)** 为置换 **f** 的循环节个数。

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

参考文献

以下参考文献或书籍乱序

1. 《算法导论（第二版）》Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein 机械工业出版社
2. 《算法艺术与信息学竞赛》 刘汝佳，黄亮 清华大学出版社
3. 《ACM/ICPC 代码库》 吉林大学计算机科学与技术学院 2005 级
4. 《ACM 小组内部预定函数 Version 2.0》IcyFenix
5. 《位运算简介及实用技巧》 北京大学 Matrix67
6. 《Preparation for NOIP2007》AiFreedom
7. 《编程之美》 电子工业出版社
8. 《数据结构（C 语言版）》 黄晓东 电子工业出版社
9. 《C 程序设计（第 3 版）》 谭浩强 清华大学出版社
10. 《左偏树的特点及其应用》 黄源河 NOI 国家集训论文
11. 《后缀数组》 芜湖一中 许智磊 NOI 国家集训论文