



南京工業大學  
NANJING TECH  
UNIVERSITY

## ICPC Template Manual



作者: 贺梦杰

June 13, 2019

# Contents

<b>1</b>	<b>基础</b>	<b>3</b>
1.1	IO 优化 . . . . .	4
<b>2</b>	<b>搜索</b>	<b>5</b>
<b>3</b>	<b>动态规划</b>	<b>6</b>
<b>4</b>	<b>字符串</b>	<b>7</b>
4.1	KMP . . . . .	8
<b>5</b>	<b>数据结构</b>	<b>10</b>
5.1	并查集 . . . . .	11
5.2	线段树 . . . . .	12
5.2.1	基础操作 . . . . .	12
5.2.2	单点更新 . . . . .	12
5.2.3	区间更新 . . . . .	13
5.2.4	区间查询 . . . . .	14
5.3	树状数组 . . . . .	15
5.3.1	单点修改, 区间查询 . . . . .	15
5.3.2	区间修改, 单点查询 . . . . .	15
5.3.3	区间修改, 区间查询 . . . . .	16
5.4	二维树状数组 . . . . .	16
5.4.1	单点修改, 区间查询 . . . . .	16
5.4.2	区间修改, 区间查询 . . . . .	17
<b>6</b>	<b>图论</b>	<b>20</b>
6.1	最小生成树 . . . . .	21
6.1.1	Prim . . . . .	21
6.1.2	Kruskal . . . . .	22
6.1.3	次小生成树 . . . . .	23
6.2	最近公共祖先 . . . . .	24

---

6.2.1	倍增 . . . . .	24
6.3	强连通分量 . . . . .	26
6.3.1	Tarjan . . . . .	26
6.3.2	缩点 DAG . . . . .	27
6.4	最短路 . . . . .	28
6.4.1	Dijkstra . . . . .	28
<b>7</b>	<b>数学</b>	<b>30</b>
7.1	快速幂 . . . . .	31
7.2	矩阵快速幂 . . . . .	31
7.3	组合数取模 . . . . .	33
<b>8</b>	<b>计算几何</b>	<b>34</b>
<b>9</b>	<b>其他</b>	<b>35</b>

# Chapter 1

## 基础

## 1.1 IO 优化

```
1 ios::sync_with_stdio(false);  
2 cin.tie(0);
```

## Chapter 2

## 搜索

## Chapter 3

## 动态规划

## Chapter 4

## 字符串



## 4.1 KMP

```
1 int *Get_next(string str)
2 {
3     int *ptr = new int[str.length()];
4     // 申请next数组
5     ptr[0] = 0;           // 首位next值为0
6     int i = 1;           // 初始化
7     int j = 0;           // 初始化
8     int len = str.length(); // 模式串长度
9     while (i < len)
10    {
11        if (str[i] == str[j])
12        {
13            ptr[i] = j + 1;
14            j++;
15            i++; // 确定前缀后缀相同的长度
16        }
17        else
18        {
19            // 不同时
20            if (j != 0)
21                j = ptr[j - 1]; // j回到前一个字符的next值位置
22            else
23            {
24                ptr[i] = 0; // 回到模式串的第一个字符
25                i++;
26            }
27        }
28    }
29    return ptr;
30 }
31 int KMP(string s, string p)
32 {
33     int *next = Get_next(p);
34     // 获得next数组
35     int i = 0;
36     int j = 0;
37     int len = s.length();
38     while (i < len)
39     {
40         if (s[i] == p[j])
41         {
```

```
42         i++;
43         j++; //匹配
44         if (j >= p.length())
45             return i - j;
46     }
47     else
48     {
49         //字符不相同回到前一个字符的next值位置
50         if (j != 0)
51             j = next[j - 1];
52         else
53             i++;
54     }
55 }
56 return -1;
57 }
```

来源: <https://www.bilibili.com/video/av47471886?from=search&seid=4651914725266859344>

## Chapter 5

## 数据结构

## 5.1 并查集

```
1  #define MAX 1010
2  struct node
3  {
4      int par;
5      //int rank;
6      //路径压缩后 rank=1或2 rank失去了意义
7      int data;
8  };
9  node ns[MAX];
10 void Init()
11 {
12     for (int i = 1; i < MAX; i++)
13     {
14         ns[i].par = i;
15     }
16 }
17 int Find(int i)
18 {
19     if (ns[i].par == i)
20     {
21         //返回根结点
22         return i;
23     }
24     ns[i].par = Find(ns[i].par);
25     //路径压缩
26     return ns[i].par;
27 }
28 void Union(int i, int j)
29 {
30     int pi = Find(i);
31     int pj = Find(j);
32     if (pi != pj)
33     {
34         ns[pi].par = pj;
35     }
36 }
```

## 5.2 线段树

### 5.2.1 基础操作

```
1  const int N = 1e5 + 10;
2  #define ls(a) (a << 1)
3  #define rs(a) (a << 1 | 1)
4  struct node
5  {
6      int val;
7      int lazy;
8  };
9  node tree[N << 2];
10 int a[N];
11 void PushUp(int rt)
12 {
13     tree[rt].val = tree[ls(rt)].val + tree[rs(rt)];
14 }
15 void PushDown(int ls, int rs, int rt)
16 {
17     tree[ls(rt)].val += ls * tree[rt].lazy;
18     tree[rs(rt)].val += rs * tree[rt].lazy;
19     tree[ls(rt)].lazy += tree[rt].lazy;
20     tree[rs(rt)].lazy += tree[rt].lazy;
21     tree[rt].lazy = 0;
22 }
23 void Build(int left, int right, int rt)
24 {
25     if (left == right)
26     {
27         tree[rt].val = a[left];
28         return;
29     }
30     int mid = (left + right) >> 1;
31     Build(left, mid, ls(rt));
32     Build(mid + 1, right, rs(rt));
33     PushUp(rt);
34     //向上更新
35 }
```

### 5.2.2 单点更新

```
1 void Update(int left, int right, int rt, int pos, int val)
2 {
3     if (left == right && left == pos)
4     {
5         tree[rt].val += val;
6         return;
7     }
8     int mid = (left + right) >> 1;
9     if (tree[rt].lazy)
10    {
11        PushDown(mid - left + 1, right - mid, rt);
12    }
13    if (mid >= pos)
14        Update(left, mid, ls(rt), pos, val);
15    else if (pos > mid)
16        Update(mid + 1, right, rs(rt), pos, val);
17    PushUp(rt);
18 }
```

例题: <https://www.luogu.org/problemnew/show/P3372>

### 5.2.3 区间更新

```
1 void Update(int left, int right, int rt, int s, int t, int val)
2 {
3     if (left >= s && right <= t)
4     {
5         tree[rt].val += (right - left + 1) * val;
6         tree[rt].lazy += val;
7         return;
8     }
9     int mid = (left + right) >> 1;
10    if (tree[rt].lazy)
11    {
12        PushDown(mid - left + 1, right - mid, rt);
13    }
14    if (mid < s)
15        Update(mid + 1, right, rs(rt), s, t, val);
16    else if (mid >= t)
17        Update(left, mid, ls(rt), s, t, val);
18    else
19    {
20        Update(left, mid, ls(rt), s, t, val);
```

```
21         Update(mid + 1, right, rs(rt), s, t, val);
22     }
23     PushUp(rt);
24 }
```

#### 5.2.4 区间查询

```
1 void Query(int left, int right, int s, int t, int rt)
2 {
3     if (left >= s && right <= t)
4     {
5         return tree[rt].val;
6     }
7     int mid = (left + right) >> 1;
8     if (tree[rt].lazy)
9         PushDown(mid - left + 1, right - mid, rt);
10    long long sum = 0;
11    if (mid < s)
12        sum += Query(mid + 1, right, rs(rt), s, t, val);
13    else if (mid >= t)
14        sum += Query(left, mid, ls(rt), s, t, val);
15    else
16    {
17        sum += Query(left, mid, ls(rt), s, t, val);
18        sum += Query(mid + 1, right, rs(rt), s, t, val);
19    }
20    return sum;
21 }
```

例题: <https://www.luogu.org/problemnew/show/P3373>

## 5.3 树状数组

推荐阅读: <https://www.cnblogs.com/RabbitHu/p/BIT.html>

### 5.3.1 单点修改, 区间查询

```
1 #define N 1000100
2 long long c[N];
3 int n,q;
4 int lowbit(int x)
5 {
6     return x&(-x);
7 }
8 void change(int x,int v)
9 {
10     while(x<=n)
11     {
12         c[x]+=v;
13         x+=lowbit(x);
14     }
15 }
16 long long getsum(int x)
17 {
18     long long ans=0;
19     while(x>=1)
20     {
21         ans+=c[x];
22         x-=lowbit(x);
23     }
24     return ans;
25 }
```

例题: <https://loj.ac/problem/130>

### 5.3.2 区间修改, 单点查询

引入差分数组来解决树状数组的区间更新

```
1 //初始化
2 change(i,cur-pre);
3 //区间修改
4 change(l,x);
5 change(r+1,-x);
6 //单点查询
```



7 getsum(x)

例题: <https://loj.ac/problem/131>

### 5.3.3 区间修改, 区间查询

```

1 //初始化
2 change(c1,i,cur-pre);
3 change(c2,i,i*(cur-pre));
4 //为什么这么写? 你需要写一下前缀和的表达式
5 //区间修改
6 change(c1,l,x);
7 change(c2,l,l*x);
8 change(c1,r+1,-x);
9 change(c2,r+1,-(r+1)*x);
10 //区间查询
11 temp1=l*getsum(c1,l-1)-getsum(c2,l-1);
12 temp2=(r+1)*getsum(c1,r)-getsum(c2,r);
13 ans=temp2-temp1

```

例题: <https://loj.ac/problem/132>

## 5.4 二维树状数组

### 5.4.1 单点修改, 区间查询

```

1 #define N 5050
2 long long tree[N][N];
3 long long n,m;
4 long long lowbit(long long x)
5 {
6     return x&(-x);
7 }
8 void change(long long x,long long y,long long val)
9 {
10     long long init_y=y;
11     //这里注意n,m的限制
12     while(x<=n)
13     {
14         y=init_y;
15         while(y<=m)
16         {
17             tree[x][y]+=val;

```

```

18         y+=lowbit(y);
19     }
20     x+=lowbit(x);
21 }
22 }
23 long long getsum(long long x,long long y)
24 {
25     long long ans=0;
26     long long init_y=y;
27     while(x>=1)
28     {
29         y=init_y;
30         while(y>=1)
31         {
32             ans+=tree[x][y];
33             y-=lowbit(y);
34         }
35         x-=lowbit(x);
36     }
37     //这里画图理解
38     return ans;
39 }
40 //初始化
41 change(x,y,k);
42 //二维前缀和
43 ans = getsum(c,d)+getsum(a-1,b-1)-getsum(a-1,d)-getsum(c,b-1);

```

例题: <https://loj.ac/problem/133>

### 5.4.2 区间修改, 区间查询

```

1  #define N 2050
2  long long t1[N][N];
3  long long t2[N][N];
4  long long t3[N][N];
5  long long t4[N][N];
6  long long n,m;
7  long long lowbit(long long x)
8  {
9      return x&(-x);
10 }
11 long long getsum(long long x,long long y)
12 {

```

```
13     long long ans=0;
14     long long init_y=y;
15     long long init_x=x;
16     while(x>=1)
17     {
18         y=init_y;
19         while(y>=1)
20         {
21             ans+=(init_x+1)*(init_y+1)*t1[x][y];
22             ans-=(init_y+1)*t2[x][y];
23             ans-=(init_x+1)*t3[x][y];
24             ans+=t4[x][y];
25             y-=lowbit(y);
26         }
27         x-=lowbit(x);
28     }
29     return ans;
30 }
31 void change(long long x,long long y,long long val)
32 {
33     long long init_x=x;
34     long long init_y=y;
35     while(x<=n)
36     {
37         y=init_y;
38         while(y<=m)
39         {
40             t1[x][y]+=val;
41             t2[x][y]+=init_x*val;
42             t3[x][y]+=init_y*val;
43             t4[x][y]+=init_x*init_y*val;
44             y+=lowbit(y);
45         }
46         x+=lowbit(x);
47     }
48 }
49 //区间修改
50 change(c+1,d+1,x);
51 change(a,b,x);
52 change(a,d+1,-x);
53 change(c+1,b,-x);
54 //区间查询
55 ans=getsum(c,d)+getsum(a-1,b-1)-getsum(c,b-1)-getsum(a-1,d);
```

例题: <https://loj.ac/problem/135>

## Chapter 6

## 图论

## 6.1 最小生成树

### 6.1.1 Prim

```
1  #define inf 0x3f3f3f3f
2  const int N = 2e5 + 20;
3  struct node
4  {
5      long long u, v, w;
6      node(int uu, int vv, int ww) : u(uu), v(vv), w(ww)
7      {
8      }
9      bool operator<(const node n) const
10     {
11         return w > n.w;
12     }
13 };
14 long long n, m;
15 priority_queue<node> q;
16 vector<pair<long long, long long>> G[N];
17 bool vis[N];
18 long long Prim()
19 {
20     long long ans = 0;
21     for (auto ele : G[1])
22     {
23         q.push(node(1, ele.first, ele.second));
24     }
25     memset(vis, 0, sizeof(vis));
26     vis[1] = 1;
27     int t = n - 1;
28     while (t--)
29     {
30         node top = q.top();
31         q.pop();
32         while (vis[top.v])
33         {
34             top = q.top();
35             q.pop();
36         }
37         ans += top.w;
38         cout<<ans<<endl;
39         vis[top.v] = 1;
```

```

40         for (auto ele : G[top.v])
41         {
42             q.push(node(top.v, ele.first, ele.second));
43         }
44     }
45     return ans;
46 }

```

### 6.1.2 Kruskal

基于并查集

```

1  const long long MAXN = 2e5 + 20;
2  struct Edge
3  {
4      long long u, v, w;
5      bool operator<(Edge e) const
6      {
7          return w > e.w;
8      }
9      Edge(long long uu, long long vv, long long ww) : u(uu), v(vv), w(ww)
10     {
11     }
12 };
13 long long fa[MAXN];
14 long long Find(long long x)
15 {
16     if (fa[x] == -1)
17         return x;
18     else
19         return fa[x] = Find(fa[x]);
20 }
21 priority_queue<Edge> q;
22 long long n; //点
23 long long m; //边
24 long long Kruskal()
25 {
26     memset(fa, -1, sizeof(fa));
27     long long cnt = 0;
28     long long ans = 0;
29     long long fu, fv;
30     while (!q.empty())

```

```
31     {
32         Edge now = q.top();
33         q.pop();
34         fu = Find(now.u);
35         fv = Find(now.v);
36         if (fu != fv)
37         {
38             //cout<<"add:"<<now.u<<" "<<now.v<<" "<<now.w<<endl;
39             ans += now.w;
40             fa[fu] = fv;
41             cnt++;
42         }
43         if (cnt == n - 1)
44             break;
45     }
46     if (cnt < n - 1)
47         return -1;
48     return ans;
49 }
```

例题: <https://loj.ac/problem/123>

### 6.1.3 次小生成树

倍增 LCA 维护最小生成树

代码过长.....

例题: <https://loj.ac/problem/10133>



## 6.2 最近公共祖先

### 6.2.1 倍增

```
1  #define N 500050
2  int depth[N];
3  int fa[N][20];
4  vector<int> v[N];
5  int lg[N]; //log2(n) floor
6  int n, m, s;
7  void init()
8  {
9      lg[0] = -1;
10     //floor
11     for (int i = 1; i <= n; i++)
12     {
13         lg[i] = lg[i >> 1] + 1;
14     }
15 }
16 void DFS(int cur, int pre)
17 {
18     depth[cur] = depth[pre] + 1;
19     fa[cur][0] = pre;
20     for (int i = 1; (1 << i) <= depth[cur]; i++)
21     {
22         fa[cur][i] = fa[fa[cur][i - 1]][i - 1];
23     }
24     for (auto ele : v[cur])
25     {
26         if (ele != pre)
27         {
28             DFS(ele, cur);
29         }
30     }
31 }
32 int LCA(int a, int b)
33 {
34     // assume depth[a]>=depth[b]
35     if (depth[a] < depth[b])
36         swap(a, b);
37     // reset to the same depth
38     while (depth[a] > depth[b])
39     {
```

```
40     a = fa[a][lg[depth[a] - depth[b]]];
41     //up
42 }
43 if (a == b)
44     return a;
45 for (int k = lg[depth[a]] + 1; k >= 0; k--)
46 {
47     if (fa[a][k] != fa[b][k])
48     {
49         a = fa[a][k];
50         b = fa[b][k];
51         //up
52     }
53 }
54 return fa[a][0];
55 }
```

例题: <https://www.luogu.org/problemnew/show/P3379>

## 6.3 强连通分量

### 6.3.1 Tarjan

```
1  const int N = 10500;
2  vector<int> G[N];
3  bool vis[N];
4  int dfn[N];
5  int low[N];
6  int cnt;
7  stack<int> s;
8  int n, m;
9  void init()
10 {
11     memset(vis, 0, sizeof(vis));
12     memset(dfn, 0, sizeof(dfn));
13     cnt = 0;
14 }
15 void DFS(int cur)
16 {
17     dfn[cur] = low[cur] = ++cnt;
18     s.push(cur);
19     vis[cur] = 1;
20     for (auto ele : G[cur])
21     {
22         if (!dfn[ele])
23         {
24             DFS(ele);
25             low[cur] = min(low[cur], low[ele]);
26         }
27         else if (vis[ele])
28         {
29             low[cur] = min(low[cur], dfn[ele]);
30         }
31     }
32     if (dfn[cur] == low[cur])
33     {
34         while (1)
35         {
36             int t = s.top();
37             s.pop();
38             vis[t] = 0;
39             if (t == cur)
```

```
40         break;
41     }
42 }
43 }
```

例题: <https://www.luogu.org/problemnew/show/P2863>

### 6.3.2 缩点 DAG

利用强连通分量缩点为有向无环图

Tarjan 加入染色

```
1  if (dfn[cur] == low[cur])
2  {
3      sum++;
4      while (1)
5      {
6          int t = s.top();
7          s.pop();
8          color[t] = sum;
9          pnum[sum]++;
10         vis[t] = 0;
11         if (t == cur)
12             break;
13     }
14 }
```

例题: <https://www.luogu.org/problemnew/show/P2341>

## 6.4 最短路

### 6.4.1 Dijkstra

非负权图，单源最短路径

$O((N + M) \log M)$

```
1  #define N 100100
2  #define ll long long
3  #define inf 2147483647
4  vector<pair<int, ll>> G[N];
5  struct edge
6  {
7      int to;
8      ll weight;
9      edge(int i, ll w) : to(i), weight(w){}
10     bool operator<(const edge &e) const{return weight > e.weight
11         ;}
12 };
13 ll dis[N];
14 bool vis[N];
15 int n, m, s;
16 void Dijkstra(int start)
17 {
18     for (int i = 1; i <= n; i++){dis[i] = inf;}
19     priority_queue<edge> q;
20     q.push(edge(start, 0));
21     dis[start] = 0;
22     while (!q.empty())
23     {
24         edge now = q.top();
25         q.pop();
26         if (!vis[now.to])
27         {
28             vis[now.to] = 1;
29             for (auto ele : G[now.to])
30             {
31                 if (!vis[ele.first] && now.weight + ele.second <
32                 dis[ele.first])
33                 {
34                     dis[ele.first] = now.weight + ele.second;
35                     q.push(edge(ele.first, dis[ele.first]));
36                 }
37             }
38         }
39     }
40 }
```

```
36         }  
37     }  
38 }
```

例题: <https://www.luogu.org/problemnew/show/P3371>

例题: <https://www.luogu.org/problemnew/show/P4779>

## Chapter 7

## 数学

## 7.1 快速幂

### 递归形式

```
1 #define mod 1000000007
2 long long quick_pow(long long a, long long b)
3 {
4     if (b == 0)
5         return 1;
6     long long temp = quick_pow(a, b >> 1);
7     if (b & 1)
8         return a % mod * temp % mod * temp % mod;
9     else
10         return temp % mod * temp % mod; //快速幂
11 }
```

### 循环形式

如果递归形式栈溢出可使用循环形式

```
1 #define mod 1000000007
2 long long quick_pow(long long x, long long n)
3 {
4     long long ret=1;
5     long long temp=x%mod;
6     while(n)
7     {
8         if(n&1)
9         {
10             ret=(ret*temp)%mod;
11         }
12         temp=(temp*temp)%mod; //偶次
13         n>>=1;
14     }
15     return ret; //结果
16 }
```

## 7.2 矩阵快速幂

```
1 #define N 105
2 int m; //矩阵阶
3 struct matrix
```



```

4 {
5     long long a[N][N];
6     matrix(){memset(a, 0, sizeof(a));}
7 };
8 matrix matrix_mul(matrix m1, matrix m2)
9 {
10     matrix ans;
11     for (int k = 0; k < m; k++)
12     {
13         for (int i = 0; i < m; i++)
14         {
15             if (m1.a[i][k]) //剪枝
16             {
17                 for (int j = 0; j < m; j++)
18                 {
19                     if (m2.a[k][j]) //剪枝
20                     {
21                         ans.a[i][j] = (ans.a[i][j] + (m1.a[i][k]
22 * m2.a[k][j]) % mod) % mod;
23                     }
24                 }
25             }
26         }
27     }
28     return ans;
29 }
30 matrix quick_pow_matrix(matrix m1, long long k)
31 {
32     matrix ans; //递归写法 可能堆栈溢出
33     for (int i = 0; i < m; i++)
34         ans.a[i][i] = 1;
35     while (k)
36     {
37         if (k & 1)
38             ans = matrix_mul(ans, m1);
39         m1 = matrix_mul(m1, m1);
40         k >>= 1;
41     }
42     return ans;
43 }

```

## 7.3 组合数取模

```
1  #define N 100010
2  #define mod 1000000007
3  #define ll long long
4  ll fac[N]; //阶乘
5  ll inv[N]; //阶乘逆元
6  void init()
7  {
8      fac[0]=1;
9      for(int i=1;i<N;i++)
10     {
11         fac[i]=(fac[i-1]*i)%mod;
12     }
13     inv[N-1]=quick_pow(fac[N-1],mod-2); //费马小定理 求逆元a^(p-1)%
//如果 ax%p=1, 那么x的最小正整数解就是 a 模p的逆元
14     //如果 ax%p=1, 那么x的最小正整数解就是 a 模p的逆元
15     for(int i=N-2;i>=0;i--)
16     {
17         inv[i]=inv[i+1]*(i+1)%mod;
18     }
19 }
20 ll C(ll a,ll b)
21 {
22     if(b>a) return 0;
23     if(b==0) return 1;
24     return fac[a]*inv[b]%mod*inv[a-b]%mod;
25 }
```

## Chapter 8

## 计算几何

## Chapter 9

### 其他