数据结构

1. 链表与邻接表

```
struct Node
{
    int val;
    Node *next;
}; //
```

⇒ new Node();

速度慢

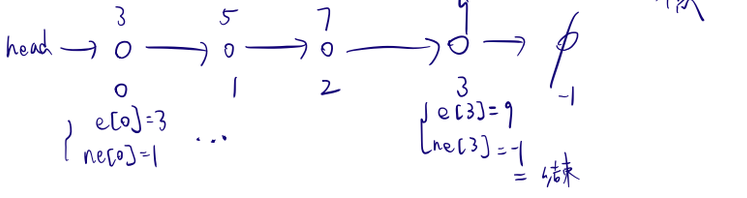
c++中new的动态操作慢

速度快

数组模拟链表 { 单链表：邻接表(1个链表) ⇒ 存储图与树
双链表：优化某些问题

val：e[N]
next：ne[N]

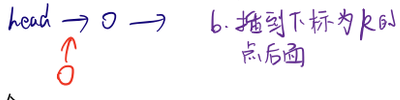
head // 头结点的下标
idx（指针）→ 当前已经用到了的最后一个点



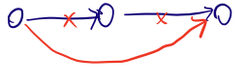
单链表

操作：① 初始化 head=-1, idx=0

② 插入



③ 删除



```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010;

int e[N],ne[N];
int head,idx;

void init()
{
    head = -1;
    idx = 1;
}

void insert_head(int num)
{
    e[idx] = num;
    ne[idx] = head;
    head = idx;
    idx ++;
}

void insert(int k,int num)
{
    e[idx] = num;
    ne[idx] = ne[k];
    ne[k] = idx;
    idx ++;

}

void deletek(int k)
{
    if(k == 0) head = ne[head];
    else ne[k] = ne[ne[k]];

}
```

```
int main()
{
    init();
    int n;
    scanf("%d", &n);
    while(n--)
    {
        string a;
        cin>>a;
        if(a == "H")
        {
            int num;
            scanf("%d", &num);
            insert_head(num);
        }
        else if(a == "I")
        {
            int k,num;
            scanf("%d%d",&k,&num);
            insert(k,num);
        }
        else if(a == "D")
        {
            int k;
            scanf("%d",&k);
            deletek(k);
        }
    }
    while(head != -1)
    {
        printf("%d ",e[head]);
        head = ne[head];
    }
    return 0;
}
```

。

双链表

定义两个 l[N], r[N]

含 head：0

tail：1

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010;

int e[N],l[N],r[N];
int idx;
// 0 左端点。1 右端点

void init()
{
    r[0] = 1;
    l[1] = 0;
    idx = 2;
}

void insert_l(int num)
{
    e[idx] = num;
    l[idx] = 0;
    r[idx] = r[0];
    l[r[0]] = idx;
    r[0] = idx;

    idx += 1;
}
void insert_r(int num)
{
    e[idx] = num;
    r[idx] = 1;
    l[idx] = l[1];
    r[l[1]] = idx;
    l[1] = idx;

    idx += 1;
}
```

```
void delete_k(int k)
{

    r[l[k]] = r[k];
    l[r[k]] = l[k];

}
void insert_lk(int k,int num)
{
    e[idx] = num;
    l[idx] = l[k];
    r[idx] = k;
    r[l[k]] = idx;
    l[k] = idx;
    idx++;
}
void insert_rk(int k,int num)
{
    e[idx] = num;
    l[idx] = k;
    r[idx] = r[k];
    l[r[k]] = idx;
    r[k] = idx;
    idx++;
}
```

```
int main()
{
    init();
    int n;
    scanf("%d", &n);
    while(n--)
    {
        string op;
        cin>>op;
        if(op == "L")
        {
            int num;
            cin>>num;
            insert_l(num);
        }
        else if(op == "R")
        {
            int num;
            cin>>num;
            insert_r(num);
        }
        else if(op == "D")
        {
            int k;
            cin>>k;
            delete_k(k+1);
        }
        else if(op == "IL")
        {
            int k,num;
            cin>>k>>num;
            insert_lk(k+1,num);
        }
        else
        {
            int k,num;
            cin>>k>>num;
            insert_rk(k+1,num);
        }
    }

    int head = 0;
    while(true)
    {
        head = r[head];
        if(head != 1) printf("%d ",e[head]);
        else break;
    }

    return 0;
}
```

邻接表

开n个单链表

(PS 图论)

2. 栈与队列

栈：先进后出

队列：先进先出

栈：

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 1e5 + 10;

int stack[N];
int head;
void init()
{
    head = -1;
}

int main()
{
    init();
    int n;
    scanf("%d", &n);
    while (n -- )
    {
        string op;
        cin>>op;
        if(op == "push")
        {
            int x;
            scanf("%d", &x);
            stack[++head] = x;
        }
        else if(op == "pop")
        {
            head --;
        }
        else if(op == "query")
        {
            printf("%d\n",stack[head]);
        }
        else
        {
            if(head==-1) printf("YES\n");
            else printf("NO\n");
        }
    }
}
```

队列：

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 1e5 + 10;

int queue[N];
int head,tail;

void init()
{
    head = -1;
    tail = -1;
}

int main()
{
    init();
    int n;
    scanf("%d", &n);
    while (n -- )
    {
        string op;
        cin>>op;
        if(op == "push")
        {
            int x;
            scanf("%d", &x);
            if(head == -1) head++;
            queue[++tail] = x;
        }
        else if(op == "pop")
        {
            head ++;
        }
        else if(op == "query")
        {
            printf("%d\n",queue[head]);
        }
        else
        {
            if(head==-1 || head>tail) printf("YES\n");
            else printf("NO\n");
        }
    }
}
```

↗ 有序且单调 !!!

单调栈：找到一个数左边离它最近且比它大的数 =(小)

3 4 2 7 5

⇒ -1 3 -1 2 2 (依次返回)

暴力 (两重循环) i 0…n) j (i-1 … 0)

(栈：存 i 左边的数)

$a_1, a_2 \cdots a_{i-1}$

若 $a_3 \geq a_5$，则 a_3 一定不会被输出

每来一个元素，从栈顶寻找：

① 如果 x < 栈顶 → 删栈顶

直到 找到首个比 x 小的数 / -1 返回

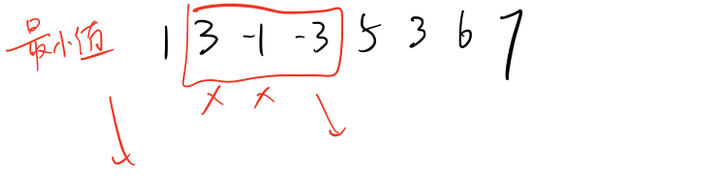
② 把入栈元素到栈顶

```
#include <iostream>
#include <cstring>
#include <algorithm>

const int N = 1e5 + 10;

int s[N];
int head=-1;

int main()
{
    int n;
    scanf("%d", &n);
    while (n -- )
    {
        int num;
        scanf("%d", &num);
        while(head >= 0 && s[head] >= num)
        {
            head --;
        }
        if(head == -1) printf("-1 ");
        else printf("%d ",s[head]);
        s[++head] = num;
    }
}
```

单调队列

经典：滑动窗口中的最大值/最小值

↘ 暴力：$O(n^2)$



队列里保证新元素最大，在队尾！

最小值时，维护的队列

① 满足在窗口内

② 满足单调上升

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 1e6 + 10;
int hh,tt=-1;

int a[N],q[N];

int main()
{
    int n,k;
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i ++ ) scanf("%d",
&a[i]);
    for (int i = 0; i < n; i ++ )
    {
        if(hh<=tt && i-k+1 >q[hh]) hh++;
        while(hh<=tt && a[q[tt]]>a[i]) tt--;
        q[++tt] = i;
        if(i>=k-1) printf("%d ",a[q[hh]]);
    }
    printf("\n");
    int hh=0,tt=-1;
     for (int i = 0; i < n; i ++ )
    {
        if(hh<=tt && i-k+1 >q[hh]) hh++;
        while(hh<=tt && a[q[tt]]<a[i]) tt--;
        q[++tt] = i;
        if(i>=k-1) printf("%d ",a[q[hh]]);
    }
}
```

KMP

1. 暴力算法
2. 优化

模式串 S，模板串 P

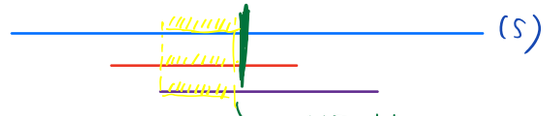
P为S的子串（多次）

求P在S中所有出现位置的起始下标

```
for(int i=1; i<=n; i++)
{
    bool flag = true;
    for(int j=1; j<=m; j++)
        if (s[i+j-1] != p[j])
        { flag = false;
          break;
        }
}
```

朴素算法

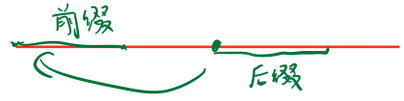
匹配过程



如此不一样

失败后，模板串向后移到什么时候才能继续匹配

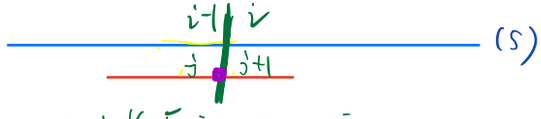
对模板串预处理！



相等的长度最大是多少？

next[i]：以i为终点的后缀

next[i] = j = 前缀 → P[1…j] = P[i-j+1,…i]



i-1都匹配，i不匹配

以j为终点的后缀，匹配前缀最长？

next[j]

求KMP数组过程

补充笔记

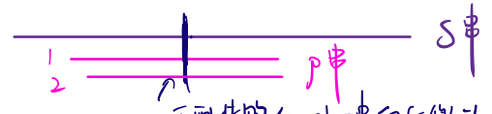
kmp：字符串匹配算法

1. s[] 模板串
2. p[] 模板串
3. 非平凡前缀：除最后一个字符外，一个字符串全部头部组合
4. 非平凡后缀：除第一个字符外，一个字符串全部的尾部组合
5. 部分匹配值：前缀 & 后缀最长共有的元素长度
6. next[] ：部分匹配值表，存储每一个下标对应的部分匹配值

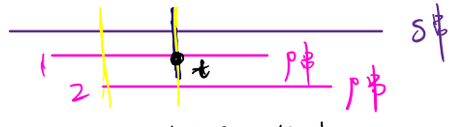
核心思想：每次匹配失败，不是把p后移一位，而是移动到下一次可以和前面部分匹配的位置

1. 暴力做法

```
for(i=1; i<=n; i++)
{  flag=true;
   for(j=1; j<=m; j++)
   {
      if(s[i+j-1] != p[j])
      {  flag=false;
         break;
      }
   }
}
```



匹配失败后，则S串向后移动1位，从头开始匹配 (1→2)



匹配失败后，移到下一次最大可匹配位置

以七结尾的字符串：前缀=后缀的最长长度 (next)

2. next 模拟

next[j]：以j为下标的后缀与前缀相同的最大长度

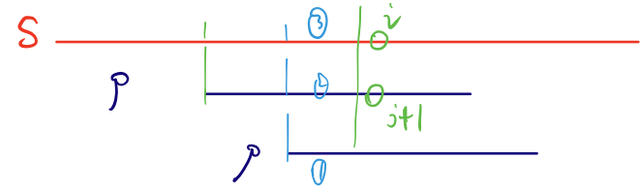
例：P串 = a b c a b
1 2 3 4 5

next[1] = 0 → 前缀 = 后缀 = ∅
next[2] = 0 → 前缀 = a 后缀 = b
next[3] = 0 → 前缀 = a, ab 后缀 = c, bc
next[4] = 1 → 前缀 = a, ab, abc 后缀 = a, ca, bca
next[5] = 2 → 前缀 = a, ab, abc, abca
后缀 = b, ab, cab, bcab

3. 具体思路

主要分两步：① 求kmp数组
② 匹配字符串

匹配字符串：S串P串都从1开始；i从1，j从0，比较S[i]&P[j+1]



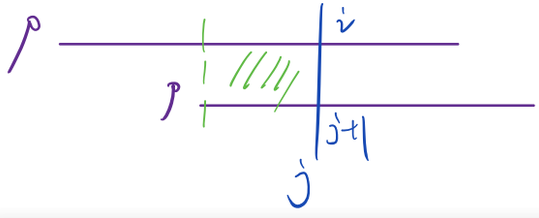
匹配中遇到 S[i] != P[j+1]
①=②=③
↓ 此时：令 j = next[j]，继续匹配下一位
直到 j = m (P长度) → 匹配成功

```
for(int i = 1, j = 0; i <= n; i++)
{
    while(j && s[i] != p[j+1]) j = ne[j];
    //如果j有对应p串的元素， 且s[i] != p[j+1], 则失配， 移动p串
    //用while是由于移动后可能仍然失配，所以要继续移动直到匹配或整个p串移到后面 (j = 0)

    if(s[i] == p[j+1]) j++;
    //当前元素匹配，j移向p串下一位
    if(j == m)
    {
        //匹配成功，进行相关操作
        j = next[j];  //继续匹配下一个子串
    }
}
```

i,j：两指针
整个P的串
下个子串 从头 还有找的时

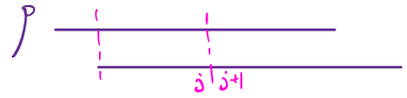
4. 求next数组的思路

模板串P与自己匹配




```
for(int i = 2, j = 0; i <= m; i++)
{
    while(j && p[i] != p[j+1]) j = next[j];

    if(p[i] == p[j+1]) j++;

    next[i] = j;
}
```

[自身匹配 不断更新next数组]



P：从2开始

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 1e6+10;
char p[N],s[N];
int ne[N];

int main()
{
    int m,n;
    cin>>n>>p+1>>m>>s+1;
    for (int i = 2,j=0; i <= n; i ++ )
    {
        while(j && p[i] != p[j+1]) j = ne[j];
        if(p[i] == p[j+1]) j++;
        ne[i] = j;
    }

    for (int i=1,j =0; i<=m;i++)
    {
        while( j && s[i]!=p[j+1]) j = ne[j];
        if(s[i] == p[j+1]) j++;
        if (j == n)
        {
            printf("%d ",i-n);
            j = ne[j];
        }
    }
}
```