**1.1.1 动态链表**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | #include <bits/stdc++.h>  struct node{ //定义链表结点  int data; //结点的值  node \*next; //单向链表，只有一个next指针  };  int main(){  int n,m; scanf("%d %d",&n,&m);  node \*head,\*p,\*now,\*prev; //定义变量  head = new node; head->data = 1; head->next=NULL; //分配第一个结点，数据置为1  now = head; //当前指针是头  for(int i=2;i<=n;i++){  p = new node; p->data = i; p->next = NULL; //p是新结点  now->next = p; //把申请的新结点连到前面的链表上  now = p; //尾指针后移一个  }  now->next = head; //尾指针指向头：循环链表建立完成  //以上是建立链表，下面是本题的逻辑和流程。**后面4种代码，逻辑流程完全一致**。  now = head, prev = head; //从第1个开始数  while((n--) >1 ){  for(int i=1;i<m;i++){ //数到m，停下  prev = now; //记录上一个位置，用于下面跳过第m个结点  now = now->next;  }  printf("%d ", now->data); //输出第m结点，带空格  prev->next = now->next; //跳过这个结点  delete now; //释放结点  now = prev->next; //新的一轮  }  printf("%d", now->data); //打印最后一个，后面不带空格  delete now; //释放最后一个结点  return 0;  } |

**1.1.2 静态链表**

**1. 用结构体数组实现单向静态链表**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | //洛谷P1996，结构体数组实现单向静态链表  #include <bits/stdc++.h>  const int N = 105; //定义静态链表的空间大小  struct node{ //单向链表  int id, nextid; //单向指针  //int data; //如有必要，定义一个有意义的数据  }nodes[N]; //定义在全局的静态分配  int main(){  int n, m; scanf("%d%d", &n, &m);  nodes[0].nextid = 1;  for(int i=1;i<=n;i++){ nodes[i].id=i; nodes[i].nextid=i+1;}  nodes[n].nextid = 1; //循环链表：尾指向头  int now = 1, prev = 1; //从第1个开始  while((n--) >1){  for(int i=1;i<m;i++){ prev = now; now = nodes[now].nextid;} //数到m停下  printf("%d ", nodes[now].id); //带空格打印  nodes[prev].nextid = nodes[now].nextid; //跳过结点now，即删除now  now = nodes[prev].nextid; //新的now  }  printf("%d", nodes[now].nextid); //打印最后一个，后面不带空格  return 0;  } |

**2. 用结构体数组实现双向静态链表**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | //洛谷P1996，结构体数组实现双向静态链表  #include <bits/stdc++.h>  const int N = 105;  struct node{ //双向链表  int id; //结点编号  //int data; //如有必要，定义一个有意义的数据  int preid, nextid; //前一个结点，后一个结点  }nodes[N];  int main(){  int n, m; scanf("%d%d", &n, &m);  nodes[0].nextid = 1;  for(int i = 1; i <= n; i++){ //建立链表  nodes[i].id = i;  nodes[i].preid = i-1; //前结点  nodes[i].nextid = i+1; //后结点  }  nodes[n].nextid = 1; //循环链表：尾指向头  nodes[1].preid = n; //循环链表：头指向尾  int now = 1; //从第1个开始  while((n--) >1){  for(int i=1;i<m;i++) now = nodes[now].nextid; //数到m，停下  printf("%d ", nodes[now].id); //打印，后面带空格  int prev = nodes[now].preid, next = nodes[now].nextid;  nodes[prev].nextid = nodes[now].nextid; //删除now  nodes[next].preid = nodes[now].preid;  now = next; //新的开始  }  printf("%d", nodes[now].nextid); //打印最后一个，后面不带空格  return 0;  } |

**3. 用一维数组实现单向静态链表**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | //洛谷P1996，一维数组实现单向静态链表  #include<bits/stdc++.h>  int nodes[150];  int main(){  int n, m; scanf("%d%d", &n, &m);  for(int i=1;i<=n-1;i++) nodes[i]=i+1; //nodes[i]的值就是下一个结点  nodes[n] = 1; //循环链表：尾指向头  int now = 1, prev = 1; //从第1个开始  while((n--) >1){  for(int i = 1; i < m; i++){ //数到m，停下  prev = now; now = nodes[now]; //下一个  }  printf("%d ", now); //带空格  nodes[prev] = nodes[now]; //跳过结点now，即删除now  now = nodes[prev]; //新的now  }  printf("%d", now); //打印最后一个，不带空格  return 0;  } |

**1.1.3 STL list**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | //洛谷P1996，STL list  #include <bits/stdc++.h>  using namespace std;  int main(){  int n, m; cin>>n>>m;  list<int>node;  for(int i=1;i<=n;i++) node.push\_back(i); //建立链表  list<int>::iterator it = node.begin();  while(node.size()>1){ //list的大小由STL自己管理  for(int i=1;i<m;i++){ //数到m  it++;  if(it == node.end()) it = node.begin(); //循环：到末尾了再回头  }  cout << \*it <<" ";  list<int>::iterator next = ++it;  if(next==node.end()) next=node.begin(); //循环链表  node.erase(--it); //删除这个结点，node.size()自动减1  it = next;  }  cout << \*it;  return 0;  } |

**1.2.1 STL queue**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | //洛谷P1540, STL queue  #include<bits/stdc++.h>  using namespace std;  int Hash[1003]={0}; //用哈希检查内存中有没有单词，hash[i]=1表示单词i在内存中  queue<int> mem; //用队列模拟内存  int main(){  int m,n; scanf("%d%d",&m,&n);  int cnt = 0; //查词典的次数  while(n--){  int en; scanf("%d",&en); //输入一个英文单词  if(!Hash[en]){ //如果内存中没有这个单词  ++cnt;  mem.push(en); //单词进队列，放到队列尾部  Hash[en]=1; //记录内存中有这个单词  while(mem.size()>m){ //内存满了  Hash[mem.front()] = 0; //从内存中去掉单词  mem.pop(); //从队头去掉  }  }  }  printf("%d\n",cnt);  return 0;  } |

**1.2.2 手写循环队列**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | //洛谷P1540, 手写循环队列  #include<bits/stdc++.h>  #define N 1003 //队列大小  int Hash[N]={0}; //用Hash检查内存中有没有单词  struct myqueue{  int data[N]; //分配静态空间  /\* 如果动态分配，这样写： int \*data; \*/  int head, rear; //队头、队尾  bool init(){ //初始化  /\*如果动态分配，这样写：  Q.data = (int \*)malloc(N \* sizeof(int)) ;  if(!Q.data) return false; \*/  head = rear = 0;  return true;  }  int size(){ return (rear - head + N) % N;} //返回队列长度  bool empty(){ //判断队列是否为空  if(size()==0) return true;  else return false;  }  bool push(int e){ //队尾插入新元素。新的rear指向下一个空的位置  if((rear + 1) % N == head ) return false; //队列满  data[rear] = e;  rear = (rear + 1) % N;  return true;  }  bool pop(int &e){ //删除队头元素，并返回它  if(head == rear) return false; //队列空  e = data[head];  head = (head + 1) % N;  return true;  }  int front(){ return data[head]; } //返回队首，但是不删除  }Q;  int main(){  Q.init(); //初始化队列  int m,n; scanf("%d%d",&m,&n);  int cnt = 0;  while(n--){  int en; scanf("%d",&en); //输入一个英文单词  if(!Hash[en]){ //如果内存中没有这个单词  ++cnt;  Q.push(en); //单词进队列，放到队列尾部  Hash[en]=1;  while(Q.size()>m){ //内存满了  int tmp; Q.pop(tmp); //删除队头  Hash[tmp] = 0; //从内存中去掉单词  }  }  }  printf("%d\n",cnt);  return 0;  } |

**1.2.3 双端队列和单调队列**

下面是洛谷P1886的代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | #include<bits/stdc++.h>  using namespace std;  const int N = 1000005;  int a[N];  deque<int>q; //队列中的数据，实际上是元素在原序列中的位置  int main(){  int n,m; scanf("%d%d",&n,&m);  for(int i=1;i<=n;i++) scanf("%d",&a[i]);  for(int i=1;i<=n;i++){ //输出最小值  while(!q.empty() && a[q.back()]>a[i]) q.pop\_back(); //去尾  q.push\_back(i);  if(i>=m){ //每个窗口输出一次  while(!q.empty() && q.front()<=i-m) q.pop\_front(); //删头  printf("%d ", a[q.front()]);  }  }  printf("\n");  while(!q.empty()) q.pop\_front(); //清空，下面再用一次  for(int i=1;i<=n;i++){ //输出最大值  while(!q.empty() && a[q.back()]<a[i]) q.pop\_back(); //去尾  q.push\_back(i);  if(i>=m){  while(!q.empty() && q.front()<=i-m) q.pop\_front(); //删头  printf("%d ", a[q.front()]);  }  }  printf("\n");  return 0;  } |

**3、单调队列与最大子序和问题**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | //hdu 1003的贪心代码  #include<bits/stdc++.h>  using namespace std;  const int INF = 0x7fffffff;  int main(){  int t; cin >> t; //测试用例个数  for(int i = 1; i <= t; i++){  int n; cin >> n;  int maxsum = -INF; //最大子序和，初始化为一个极小负数  int start=1, end=1, p=1; //起点，终点，扫描位置  int sum = 0; //子序和  for(int j = 1; j <= n; j++){  int a; cin >> a; sum += a; //读入一个元素，累加  if(sum > maxsum){ maxsum = sum; start = p; end = j;}  if(sum < 0){ //扫到j时，若前面的最大子序和是负数，从下一个重新开始求和  sum = 0;  p = j+1;  }  }  printf("Case %d:\n",i); printf("%d %d %d\n", maxsum,start,end);  if(i != t) cout << endl;  }  return 0;  } |

题解2：动态规划DP。定义状态dp[i]，表示以第a[i]为结尾的最大子序和。dp[i]的计算有两种情况：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | // hdu 1003的DP代码  #include<bits/stdc++.h>  using namespace std;  int dp[100005]; //dp[i]: 以第i个数为结尾的最大值  int main(){  int t; cin>>t;  for(int k=1;k<=t;k++){  int n; cin >> n;  for(int i=1;i<=n;i++) cin >> dp[i]; //就用dp[]存数据a[]  int start=1, end=1, p=1; //起点，终点，扫描位置  int maxsum = dp[1];  for(int i=2; i<=n; i++){  if(dp[i-1]+dp[i] >= dp[i]) //转移方程dp[i]=max(dp[i-1]+a[i], a[i]);  dp[i] = dp[i-1]+dp[i]; // dp[i-1]+a[i]比a[i]大  else p = i; // a[i] 更大，那么dp[i]就是a[i]  if(dp[i]> maxsum ) { //dp[i]是一个更大的子序和  maxsum = dp[i]; start = p; end = i; //以p为开始, 以i为结尾  }  }  printf("Case %d:\n",k); printf("%d %d %d\n", maxsum,start,end);  if(k != t) cout << endl;  }  } |

下面是hdu1003的代码。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include<bits/stdc++.h>  using namespace std;  deque<int> dq;  int s[100005];  int main(){  int n,m; scanf("%d%d",&n,&m);  for(int i=1;i<=n;i++) scanf("%lld",&s[i]);  for(int i=1;i<=n;i++) s[i]=s[i]+s[i-1]; //计算前缀和  int ans = -1e8;  dq.push\_back(0);  for(int i=1;i<=n;i++) {  while(!dq.empty() && dq.front()<i-m) dq.pop\_front(); //队头超过m范围：删头  if(dq.empty()) ans = max(ans,s[i]);  else ans = max(ans,s[i]-s[dq.front()]); //队头就是最小的s[k]  while(!dq.empty() && s[dq.back()]>=s[i]) dq.pop\_back();//队尾大于s[i]，去尾  dq.push\_back(i);  }  printf("%d\n",ans);  return 0;  } |

**1.3.1 STL stack**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include<bits/stdc++.h>  using namespace std;  int main(){  int n; scanf("%d",&n); getchar();  while(n--){  stack<char> s;  while(true){  char ch = getchar(); //一次读入一个字符  if(ch==' '||ch=='\n'||ch==EOF){  while(!s.empty()){ printf("%c",s.top()); s.pop();} //输出并清除栈顶  if(ch=='\n'||ch==EOF) break;  printf(" ");  }  else s.push(ch); //入栈  }  printf("\n");  }  return 0;  } |

**1.3.2 手写栈**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | // hdu 1062，手写栈  #include<bits/stdc++.h>  const int N = 100100;  struct mystack{  char a[N]; //存放栈元素，字符型  int t = 0; //栈顶位置  void push(char x){ a[++t] = x; } //送入栈  char top() { return a[t]; } //返回栈顶元素  void pop() { t--; } //弹出栈顶  int empty() { return t==0?1:0;} //返回1表示空  }st;  int main(){  int n; scanf("%d",&n); getchar();  while(n--){  while(true){  char ch = getchar(); //一次读入一个字符  if(ch==' '||ch=='\n'||ch==EOF){  while(!st.empty()){ printf("%c",st.top()); st.pop();} //输出并清除栈顶  if(ch=='\n'||ch==EOF) break;  printf(" ");  }  else st.push(ch); //入栈  }  printf("\n");  }  return 0;  } |

**1.3.3 单调栈**

（1）STL stack

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include<bits/stdc++.h>  using namespace std;  int h[100001], ans[100001];  int main(){  int n; scanf("%d",&n);  for (int i=1;i<=n;i++) scanf("%d",&h[i]);  stack<int>st;  for (int i=n;i>=1;i--){  while (!st.empty() && h[st.top()] <= h[i])  st.pop(); //栈顶奶牛没我高，弹出它，直到栈顶奶牛更高为止  if (st.empty()) ans[i]=0; //栈空，没有仰望对象  else ans[i]=st.top(); //栈顶奶牛更高，是仰望对象  st.push(i); //进栈  }  for (int i=1;i<=n;i++) printf("%d\n",ans[i]);  return 0;  } |

（2）手写栈

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | #include<bits/stdc++.h>  using namespace std;  const int N = 100100;  struct mystack{  int a[N]; //存放栈元素，**int型**  int t = 0; //栈顶位置  void push(int x){ a[++t] = x; } //送入栈  int top() { return a[t]; } //返回栈顶元素  void pop() { t--; } //弹出栈顶  int empty() { return t==0?1:0;} //返回1表示空  }st;  int h[N], ans[N];  int main(){  int n; scanf("%d",&n);  for (int i=1;i<=n;i++) scanf("%d",&h[i]);  for (int i=n;i>=1;i--){  while (!st.empty() && h[st.top()] <= h[i])  st.pop(); //栈顶奶牛没我高，弹出它，直到栈顶奶牛更高  if (st.empty()) ans[i]=0; //栈空，没有仰望对象  else ans[i]=st.top(); //栈顶奶牛更高，是仰望对象  st.push(i);  }  for (int i=1;i<=n;i++) printf("%d\n",ans[i]);  return 0;  } |

**1.4.3 哈夫曼树和哈夫曼编码**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include <cstdio>  #include <iostream>  #include <algorithm>  #include <queue>  using namespace std;  int main(){  priority\_queue <int, vector<int>, greater<int> > q;  string s;  while(getline(cin, s) && s != "END"){  sort(s.begin(), s.end());  int num = 1; //一种字符出现的次数  for(int i = 1; i <= s.length(); i++){  if(s[i] != s[i-1]){ q.push(num); num = 1;}  else num++;  }  int ans = 0;  if(q.size() == 1) //题目的一个坑：只有一种字符的情况  ans = s.length();  while(q.size() > 1){ //最后一次合并不用加到ans里  int a = q.top(); q.pop(); //贪心：取出频次最少的两个  int b = q.top(); q.pop();  q.push(a+b); //把两个最小的合并成新的结点，重新放进队列  ans += a+b; //一种字符进几次队列，就累加几次。  //进一次队列，表示它在二叉树上深了一层，编码长度加1  }  q.pop();  printf("%d %d %.1f\n",s.length()\*8,ans,(double)s.length()\*8/(double)ans);  }  return 0;  } |

**1.5.3 二叉堆的手写代码**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | #include<bits/stdc++.h>  using namespace std;  const int N = 1e6 + 5;  int heap[N], len=0; //len记录当前二叉树的长度  void push(int x) { //上浮，插入新元素  heap[++len] = x;  int i = len;  while (i > 1 && heap[i] < heap[i/2]){  swap(heap[i], heap[i/2]);  i = i/2;  }  }  void pop() { //下沉，删除堆头，调整堆  heap[1] = heap[len--]; //根结点替换为最后一个结点，然后结点数量减1  int i = 1;  while ( 2\*i <= len) { //至少有左儿子  int son = 2\*i; //左儿子  if (son < len && heap[son + 1] < heap[son])  son++; //son<len表示有右儿子，选儿子中较小的  if (heap[son] < heap[i]){ //与小的儿子交换  swap(heap[son], heap[i]);  i = son; //下沉到儿子处  }  else break; //如果不比儿子小，就停止下沉  }  }  int main() {  int n; scanf("%d",&n);  while(n--){  int op; scanf("%d",&op);  if (op == 1) { int x; scanf("%d",&x); push(x); } //加入堆  else if (op == 2) printf("%d\n", heap[1]); //打印堆头  else pop(); //删除堆头  }  return 0;  } |

**1.5.4 堆和priority\_queue**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #include<bits/stdc++.h>  using namespace std;  priority\_queue<int ,vector<int>,greater<int> >q; //定义堆  int main(){  int n; scanf("%d",&n);  while(n--) {  int op; scanf("%d",&op);  if(op==1) { int x; scanf("%d",&x); q.push(x); }  else if(op==2) printf("%d\n",q.top());  else q.pop();  }  return 0;  } |