Contents

1 Basic

1.1 ascii

int	char	int	char	int	char
32		64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

1.2 limits

```
[Type]            [size]       [range]
char                1         127 to -128
signed char         1         127 to -128
unsigned char       1         0 to 255
short               2         32767 to -32768
int                 4         2147483647 to -2147483648
unsigned int        4         0 to 4294967295
long                4         2147483647 to -2147483648
unsigned long       4         0 to 18446744073709551615
long long           8
           9223372036854775807 to -9223372036854775808
double              8      1.79769e+308 to 2.22507e-308
long double         16    1.18973e+4932 to 3.3621e-4932
float               4        3.40282e+38 to 1.17549e-38
unsigned long long  8         0 to 18446744073709551615
string              32
```

2 字串

2.1 最長迴文子字串

```cpp
#include <bits/stdc++.h>
#define T(x) ((x) % 2 ? s[(x) / 2] : '.')
using namespace std;

string s;
int n;

int ex(int l, int r) {
  int i = 0;
  while(l - i >= 0 && r + i < n && T(l - i) == T(r + i)) i++;
  return i;
}

int main() {
  cin >> s;
  n = 2 * s.size() + 1;

  int mx = 0;
  int center = 0;
  vector<int> r(n);
  int ans = 1;
  r[0] = 1;
  for(int i = 1; i < n; i++) {
    int ii = center - (i - center);
    int len = mx - i + 1;
    if(i > mx) {
      r[i] = ex(i, i);
      center = i;
      mx = i + r[i] - 1;
    } else if(r[ii] == len) {
      r[i] = len + ex(i - len, i + len);
      center = i;
      mx = i + r[i] - 1;
    } else {
      r[i] = min(r[ii], len);
    }
    ans = max(ans, r[i]);
  }

  cout << ans - 1 << "\n";
  return 0;
}
```

3 STL

3.1 priority_queue

```
priority_queue：
    優先隊列，資料預設由大到小排序，即優先權高的資料會先被宣告：
    priority_queue <int> pq;
把元素 x 加進 priority_queue：
    pq.push(x);
讀取優先權最高的值：
    x = pq.top();
    pq.pop();                   //讀取後刪除
判斷是否為空的priority_queue：
    pq.empty()                  //回傳true
    pq.size()                   //回傳0
如需改變priority_queue的優先權定義：
    priority_queue<T> pq;       //預設由大到小
    priority_queue<T, vector<T>, greater<T> > pq;
                                //改成由小到大
    priority_queue<T, vector<T>, cmp> pq;    //cmp
```

3.2 queue

```
queue：佇列，資料有「先進先出」 (first in first out, FIFO)的特性。
就像排隊買票一樣，先排隊的客戶被服務。
宣告：
    queue <int> q;
把元素 x 加進 queue：
    q.push(x);
取值：
    x = q.front(); //頭
    x = q.back();  //尾
移除已經讀取的值：
    q.pop();
判斷是否為空的queue：
    q.empty() 回傳true
    q.size() 回傳零

#include <iostream>
#include <queue>
using namespace std;

int main() {
    int n;
    while (cin >> n){
        if (n == 0) break;
        queue <int> q;
        for (int i = 0; i < n; i++){
            q.push(i+1);
        }
        cout << "Discarded cards:";
        for (int i = 0; i < n-1; i++){
            if (i != 0) cout << ',';
            cout << ' ' << q.front();
            q.pop();
            q.push(q.front());
            q.pop();
        }
        cout << endl << "Remaining card: " <<
            q.front() << endl;
    }
}
```

3.3 deque

```
deque 是 C++ 標準模板函式庫 (Standard Template Library, STL)
    中的雙向佇列容器 (Double-ended Queue) ，跟 vector 相似，
    不過在 vector
        中若是要添加新元素至開端，其時間複雜度為 O(N)，
        但在 deque 中則是 O(1)。同樣地，
        也能在我們需要儲存更多元素的時候自動擴展空間，
        讓我們不必煩惱佇列長度的問題。
dq.push_back()  //在 deque 的最尾端新增元素
dq.push_front() //在 deque 的開頭新增元素
dq.pop_back()   //移除 deque 最尾端的元素
dq.pop_front()  //移除 deque 最開頭的元素
dq.back()       //取出 deque 最尾端的元素
dq.front()      //回傳 deque 最開頭的元素
dq.insert()
dq.insert(position, n, val)
    position: 插入元素的 index 值
    n: 元素插入次數
    val: 插入的元素值
dq.erase()
    //刪除元素，需要使用迭代器指定刪除的元素或位置，
    同時也會返回指向刪除元素下一元素的迭代器。
dq.clear()      //清空整個 deque 佇列。
dq.size()       //檢查 deque 的尺寸
dq.empty()      //如果 deque 佇列為空返回 1；
    若是存在任何元素，則返回0
dq.begin()      //返回一個指向 deque 開頭的迭代器
dq.end()        //指向 deque 結尾，
                    不是最後一個元素，
                    而是最後一個元素的下一個位置
```

3.4 map

```
map：存放 key-value pairs 的映射資料結構，會按 key 由小到大排序。
元素存取
operator[]：存取指定的[i]元素的資料

迭代器
begin()：回傳指向map頭部元素的迭代器
end()：回傳指向map末尾的迭代器
rbegin()：回傳一個指向map尾部的反向迭代器
rend()：回傳一個指向map頭部的反向迭代器

遍歷整個map時，利用iterator操作：
取key：it->first 或 (*it).first
取value：it->second 或 (*it).second

容量
empty()：檢查容器是否為空，空則回傳true
size()：回傳元素數量
max_size()：回傳可以容納的最大元素個數

修改器
clear()：刪除所有元素
insert()：插入元素
erase()：刪除一個元素
swap()：交換兩個map

查找
count()：回傳指定元素出現的次數
find()：查找一個元素

//實作範例
#include <bits/stdc++.h>
using namespace std;

int main(){

    //declaration container and iterator
    map<string, string> mp;
    map<string, string>::iterator iter;
    map<string, string>::reverse_iterator iter_r;

    //insert element
```

```
    mp.insert(pair<string, string>("r000",
        "student_zero"));

    mp["r123"] = "student_first";
    mp["r456"] = "student_second";

    //traversal
    for(iter = mp.begin(); iter != mp.end(); iter++)
        cout<<iter->first<<" "<<iter->second<<endl;
    for(iter_r = mp.rbegin(); iter_r != mp.rend();
        iter_r++)
        cout<<iter_r->first<<"
            "<<iter_r->second<<endl;

    //find and erase the element
    iter = mp.find("r123");
    mp.erase(iter);

    iter = mp.find("r123");

    if(iter != mp.end())
       cout<<"Find, the value is
           "<<iter->second<<endl;
    else
       cout<<"Do not Find"<<endl;

    return 0;
}

//map統計數字
#include<bits/stdc++.h>
using namespace std;

int main(){
  ios::sync_with_stdio(0),cin.tie(0);
  long long n,x;
  cin>>n;
  map <int,int> mp;
  while(n--){
    cin>>x;
    ++mp[x];
  }
  for(auto i:mp) cout<<i.first<<" "<<i.second<<endl;
}
```

3.5 unordered_map

```
unordered_map：存放 key-value pairs
    的「無序」映射資料結構。
用法與map相同
```

3.6 set

```
set： 集合，去除重複的元素，資料由小到大排序。
宣告：
    set <int> st;
把元素 x 加進 set：
    st.insert(x);
檢查元素 x 是否存在 set 中：
    st.count(x);
刪除元素 x：
    st.erase(x); // 可傳入值或iterator
清空集合中的所有元素：
    st.clear();
取值： 使用iterator
    x = *st.begin();
            // set中的第一個元素(最小的元素)。
    x = *st.rbegin();
            // set中的最後一個元素(最大的元素)。
判斷是否為空的set：
    st.empty() 回傳true
    st.size() 回傳零
常用來搭配的member function：
    st.count(x);
    auto it = st.find(x);
        // binary search, O(log(N))
    auto it = st.lower_bound(x);
        // binary search, O(log(N))
    auto it = st.upper_bound(x);
        // binary search, O(log(N))
```

3.7 multiset

```
與 set 用法雷同，但會保留重複的元素，
    資料由小到大排序。
宣告：
    multiset<int> st;
刪除資料：
    st.erase(val); 會刪除所有值為 val 的元素。
    st.erase(st.find(val)); 只刪除第一個值為 val
        的元素。
```

3.8 unordered_set

```
unordered_set 的實作方式通常是用雜湊表(hash table)，
資料插入和查詢的時間複雜度很低，為常數級別O(1)，
相對的代價是消耗較多的記憶體，空間複雜度較高，
無自動排序功能。

初始化
unordered_set<int> myunordered_set{1, 2, 3, 4, 5};

陣列初始化
int arr[] = {1, 2, 3, 4, 5};
unordered_set<int> myunordered_set(arr, arr+5);

插入元素
unordered_set<int> myunordered_set;
myunordered_set.insert(1);

迴圈遍歷 unordered_set 容器
#include <iostream>
#include <unordered_set>
using namespace std;

int main() {
    unordered_set<int> myunordered_set = {3, 1};
    myunordered_set.insert(2);
    myunordered_set.insert(5);
    myunordered_set.insert(4);
    myunordered_set.insert(5);
    myunordered_set.insert(4);

    for (const auto &s : myunordered_set) {
        cout << s << " ";
    }
    cout << "\n";

    return 0;
}

/*
output
4 5 2 1 3
*/

unordered_set 刪除指定元素
#include <iostream>
#include <unordered_set>

int main() {
    unordered_set<int> myunordered_set{2, 4, 6, 8};

    myunordered_set.erase(2);
```

```
    for (const auto &s : myunordered_set) {
        cout << s << " ";
    }
    cout << "\n";

    return 0;
}
/*
output
8 6 4
*/

清空 unordered_set 元素
unordered_set<int> myunordered_set;
myunordered_set.insert(1);
myunordered_set.clear();

unordered_set 判斷元素是否存在
unordered_set<int> myunordered_set;
myunordered_set.insert(2);
myunordered_set.insert(4);
myunordered_set.insert(6);
cout << myunordered_set.count(4) << "\n"; // 1
cout << myunordered_set.count(8) << "\n"; // 0

判斷 unordered_set 容器是否為空
#include <iostream>
#include <unordered_set>

int main() {
    unordered_set<int> myunordered_set;
    myunordered_set.clear();

    if (myunordered_set.empty()) {
        cout << "empty\n";
    } else {
        cout << "not empty, size is "<<
            myunordered_set.size() <<"\n";
    }

    return 0;
}
```

3.9 單調隊列

```
//單調隊列
"如果一個選手比你小還比你強，你就可以退役了。"--單調隊列

example 1

給出一個長度為 n 的數組，
編程輸出每 k 個連續的數中的最大值和最小值。
//寫法一
#include <bits/stdc++.h>
#define maxn 1000100
using namespace std;
int q[maxn], a[maxn];
int n, k;

void getmin() {
    // 得到這個隊列裡的最小值，直接找到最後的就行了
    int head = 0, tail = 0;
    for (int i = 1; i < k; i++) {
        while (head <= tail && a[q[tail]] >= a[i])
            tail--;
        q[++tail] = i;
    }
    for (int i = k; i <= n; i++) {
        while (head <= tail && a[q[tail]] >= a[i])
            tail--;
        q[++tail] = i;
        while (q[head] <= i - k) head++;
        cout<<a[q[head]]<<" ";
    }
}

void getmax() {  // 和上面同理
    int head = 0, tail = 0;
    for (int i = 1; i < k; i++) {
      while (head <= tail && a[q[tail]] <= a[i])
          tail--;
      q[++tail] = i;
    }
    for (int i = k; i <= n; i++) {
        while (head <= tail && a[q[tail]] <= a[i])
            tail--;
        q[++tail] = i;
        while (q[head] <= i - k) head++;
        cout<<a[q[head]]<<" ";
    }
}

int main() {
    cin>>n>>k;  //每k個連續的數
    for (int i = 1; i <= n; i++) cin>>a[i];
    getmin();
    cout<<'\n';
    getmax();
    cout<<'\n';
    return 0;
}

//寫法2
#include <iostream>
#include <cstring>
#include <deque>
using namespace std;
int a[1000005];

int main() {
    ios_base::sync_with_stdio(0);
    int n, k;
    while(cin>>n>>k) {
        for(int i=0; i<n; i++) cin >> a[i];
        deque<int> dq;
        for(int i=0; i<n; i++){
            while(dq.size() && dq.front()<=i-k)
                dq.pop_front();
            while(dq.size() && a[dq.back()]>a[i])
                dq.pop_back();
            dq.push_back(i);
            if(i==k-1) cout<<a[dq.front()];
            if(i>k-1) cout<<' '<<a[dq.front()];
        }
        if(k>n) cout<<a[dq.front()];
        cout<<'\n';
        while(dq.size()) dq.pop_back();
        for(int i=0; i<n; i++){
            while(dq.size() && dq.front()<=i-k)
                dq.pop_front();
            while(dq.size() && a[dq.back()]<a[i])
                dq.pop_back();
            dq.push_back(i);
            if(i==k-1) cout<<a[dq.front()];
            if(i>k-1)  cout<<' '<<a[dq.front()];
        }
        if(k>n) cout<<a[dq.front()];
        cout<<'\n';
    }
    return 0;
}


example 2

一個含有 n 項的數列，求出每一項前的 m
    個數到它這個區間內的最小值。
若前面的數不足 m 項則從第 1
    個數開始，若前面沒有數則輸出 0

#include<bits/stdc++.h>
using namespace std;
```

```cpp
#define re register int
#define INF 0x3f3f3f3f
#define ll long long
#define maxn 2000009
#define maxm
inline ll read() {
    ll x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9'){
    if(ch=='-') f=-1;
    ch=getchar();
  }
    while(ch>='0'&&ch<='9'){
    x=(x<<1)+(x<<3)+(ll)(ch-'0');
    ch=getchar();
  }
    return x*f;
}
int n,m,k,tot,head,tail;
int a[maxn],q[maxn];
int main() {
    n=read(), m=read();
    for(int i=1;i<=n;i++) a[i]=read();
    head=1,tail=0;//起始位置為1
        因為插入是q[++tail]所以要初始化為0
    for(int i=1;i<=n;i++)
      //每次隊首的元素就是當前的答案
    {
        cout<<a[q[head]]<<endl;
        while(i-q[head]+1>m&&head<=tail)//維護隊首
            head++;
        while(a[i]<a[q[tail]]&&head<=tail)//維護隊尾
            tail--;
        q[++tail]=i;
    }
    return 0;
}
```

4 sort

4.1 big number sort

```python
#python大數排序

while True:
  try:
    n = int(input())            # 有幾筆數字需要排序
    arr = []                    # 建立空串列
    for i in range(n):
      arr.append(int(input())) # 依序將數字存入串列
    arr.sort()                  # 串列排序
    for i in arr:
      print(i)              # 依序印出串列中每個項目
  except:
    break
```

4.2 bubble sort

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  int n;
  cin>>n;
  int a[n], tmp;
  for(int i=0; i<n; i++) cin>>a[i];
  for(int i=n-1; i>0; i--) {
    for(int j=0; j<=i-1; j++) {
      if( a[j]>a[j+1]) {
        tmp=a[j];
        a[j]=a[j+1];
        a[j+1]=tmp;
      }
    }
  }
  for(int i=0; i<n; i++) cout<<a[i]<<" ";
}
```

5 math

5.1 prime factorization

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
  int n;
  while(true) {
    cin>>n;
    for(int x=2; x<=n; x++) {
      while(n%x==0) {
        cout<<x<<"*";
        n/=x;
      }
    }
    cout<<"\b \n";
  }
  system("pause");
  return 0;
}
```

5.2 快速冪

```cpp
計算a^b
#include <iostream>
#define ll long long
using namespace std;

const ll MOD = 1000000007;
ll fp(ll a, ll b) {
  int ans = 1;
  while(b > 0) {
    if(b & 1) ans = ans * a % MOD;
    a = a * a % MOD;
    b >>= 1;
  }
  return ans;
}

int main() {
  int a, b;
  cin>>a>>b;
  cout<<fp(a,b);
}
```

6 algorithm

6.1 basic

```
min： 取最小值。
min(a, b)
min(list)
max： 取最大值。
max(a, b)
max(list)
min_element： 找尋最小元素
min_element(first, last)
max_element： 找尋最大元素
max_element(first, last)
```

```
sort： 排序，預設由小排到大。
sort(first, last)
sort(first, last, comp)： 可自行定義比較運算子 Comp 。
find： 尋找元素。
find(first, last, val)
lower_bound： 尋找第一個小於 x
    的元素位置，如果不存在，則回傳 last 。
lower_bound(first, last, val)
upper_bound： 尋找第一個大於 x
    的元素位置，如果不存在，則回傳 last 。
upper_bound(first, last, val)
next_permutation：
    將序列順序轉換成下一個字典序，如果存在回傳 true
    ，反之回傳 false 。
next_permutation(first, last)
prev_permutation：
    將序列順序轉換成上一個字典序，如果存在回傳 true
    ，反之回傳 false 。
prev_permutation(first, last)
```

6.2 binarysearch

```cpp
#include <bits/stdc++.h>
using namespace std;

int binary_search(vector<int> &nums, int target) {
    int left=0, right=nums.size()-1;
    while(left<=right){
        int mid=(left+right)/2;
        if (nums[mid]>target) right=mid-1;
        else if(nums[mid]<target) left=mid+1;
        else return mid+1;
    }
    return 0;
}

int main() {
  int n, k, x;
  cin >> n >> k;
  int a[n];
  vector<int> v;
  for(int i=0 ; i<n ; i++){
    cin >> x;
    v.push_back(x);
  }
  for(int i=0 ; i<k ; i++) cin >> a[i];
  for(int i=0 ; i<k ; i++){
    cout << binary_search(v, a[i]) << endl;
  }
}

lower_bound(a, a + n, k);      //最左邊 ≥ k 的位置
upper_bound(a, a + n, k);      //最左邊 > k 的位置
upper_bound(a, a + n, k) - 1; //最右邊 ≤ k 的位置
lower_bound(a, a + n, k) - 1; //最右邊 < k 的位置
(lower_bound, upper_bound) //等於 k 的範圍
equal_range(a, a+n, k);

/*
input
5 5
1 3 4 7 9
3 1 9 7 -2
*/

/*
output
2
1
5
4
0
*/
```

6.3 prefix sum

```cpp
// 前綴和
陣列前n項的和。
b[i] = a[0] + a[1] + a[2] + … + a[i]
區間和 [l, r]：b[r]-b[l-1] (要保留b[l]所以-1)

#include <bits/stdc++.h>
using namespace std;
int main(){
    int n;
    cin >> n;
    int a[n], b[n];
    for(int i=0; i<n; i++) cin >> a[i];
    b[0] = a[0];
    for(int i=1; i<n; i++) b[i] = b[i-1] + a[i];
    for(int i=0;i<n;i++) cout<<b[i]<<' ';
    cout<<'\n';
    int l, r;
    cin >> l >> r;
    cout << b[r] - b[l-1] ; //區間和
}
```

6.4 差分

```cpp
// 差分
用途：在區間 [l, r] 加上一個數字v。
b[l] += v; (b[0~l] 加上v)
b[r+1] -= v; (b[r+1~n] 減去v (b[r] 仍保留v) )
給的 a[] 是前綴和數列，建構 b[]，
因為 a[i] = b[0] + b[1] + b[2] + … + b[i]，
所以 b[i] = a[i] - a[i-1]。
在 b[l] 加上 v，b[r+1] 減去 v，
最後再從 0 跑到 n 使 b[i] += b[i-1]。
這樣一來，b[] 是一個在某區間加上v的前綴和。

#include <bits/stdc++.h>
using namespace std;
int a[1000], b[1000];
//a: 前綴和數列, b: 差分數列
int main(){
    int n, l, r, v;
    cin >> n;
    for(int i=1; i<=n; i++){
        cin >> a[i];
        b[i] = a[i] - a[i-1]; //建構差分數列
    }
    cin >> l >> r >> v;
    b[l] += v;
    b[r+1] -= v;

    for(int i=1; i<=n; i++){
        b[i] += b[i-1];
        cout << b[i] << ' ';
    }
}
```

6.5 greedy

```cpp
//貪心
貪心演算法的核心為，
採取在目前狀態下最好或最佳（即最有利）的選擇。
貪心演算法雖然能獲得當前最佳解，
但不保證能獲得最後（全域）最佳解，
提出想法後可以先試圖尋找有沒有能推翻原本的想法的反例，
確認無誤再實作。

//problem
有一個 N×1 的稻田，有些稻田現在有種植作物，
為了避免被動物破壞，需要放置稻草人，
```

稻草人可以保護該塊稻田和左右兩塊稻田，
請問最少需要多少稻草人才能保護所有稻田?

```
//solutoin
從左到右掃描稻田，如果第 i 塊稻田有作物，
就把稻草人放到第 i+1 塊稻田，
這樣能保護第 i,i+1,i+2 塊稻田，
接著從第 i+3 塊稻田繼續掃描。
```

```
//code
#include <bits/stdc++.h>
using namespace std;
int main(){
    string s;
    int i, n, t, tc = 1;
    cin >> t;
    while (t--){
        cin >> n >> s;
        int nc = 0;
        for (i = 0; i < n; i++)
            if (s[i] == '.')i += 2, nc++;
        cout << "Case " << tc++ << ": " << nc << endl;
    }
}
```

```
//problem
給定 N 個數，每次將兩個數 a,b 合併成 a+b，
只到最後只剩一個數，合併成本為兩數和，問最小合併成本為多少
```

7 graph

7.1 graph

```
#include<bits/stdc++.h>
using namespace std;

class Node {
public:
    int val;
    vector<Node*> children;

    Node() {}

    Node(int _val) {
        val = _val;
    }

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};

struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x),
        next(next) {}
};

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr),
        right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr),
        right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right)
        : val(x), left(left), right(right) {}
};

class ListProblem {
    vector<int> nums={};
public:
    void solve() {
        return;
    }

    ListNode* buildList(int idx) {
        if(idx == nums.size()) return NULL;
        ListNode *current=new
            ListNode(nums[idx++],current->next);
        return current;
    }

    void deleteList(ListNode* root) {
        if(root == NULL) return;
        deleteList(root->next);
        delete root;
        return;
    }
};

class TreeProblem {
    int null = INT_MIN;
    vector<int> nums = {}, result;
public:
    void solve() {

        return;
    }

    TreeNode* buildBinaryTreeUsingDFS(int left, int
        right) {
        if((left > right) || (nums[(left+right)/2] ==
            null)) return NULL;
        int mid = (left+right)/2;
        TreeNode* current = new TreeNode(
            nums[mid],
            buildBinaryTreeUsingDFS(left,mid-1),
            buildBinaryTreeUsingDFS(mid+1,right));
        return current;
    }

    TreeNode* buildBinaryTreeUsingBFS() {
        int idx = 0;
        TreeNode* root = new TreeNode(nums[idx++]);
        queue<TreeNode*> q;
        q.push(root);
        while(idx < nums.size()) {
            if(nums[idx] != null) {
                TreeNode* left = new
                    TreeNode(nums[idx]);
                q.front()->left = left;
                q.push(left);
            }
            idx++;
            if((idx < nums.size()) && (nums[idx] !=
                null)) {
                TreeNode* right = new
                    TreeNode(nums[idx]);
                q.front()->right = right;
                q.push(right);
            }
            idx++;
            q.pop();
        }
        return root;
    }

    Node* buildNAryTree() {
        int idx = 2;
        Node *root = new Node(nums.front());
        queue<Node*> q;
        q.push(root);
        while(idx < nums.size()) {
```

```
107             while((idx < nums.size()) && (nums[idx]
                    != null)) {
108                 Node *current = new Node(nums[idx++]);
109                 q.front()->children.push_back(current);
110                 q.push(current);
111             }
112             idx++;
113             q.pop();
114         }
115         return root;
116     }
117 
118     void deleteBinaryTree(TreeNode* root) {
119         if(root->left != NULL)
                deleteBinaryTree(root->left);
120         if(root->right != NULL)
                deleteBinaryTree(root->right);
121         delete root;
122         return;
123     }
124 
125     void deleteNAryTree(Node* root) {
126         if(root == NULL) return;
127         for(int i=0; i<root->children.size(); i++) {
128             deleteNAryTree(root->children[i]);
129             delete root->children[i];
130         }
131         delete root;
132         return;
133     }
134 
135     void inorderTraversal(TreeNode* root) {
136         if(root == NULL) return;
137         inorderTraversal(root->left);
138         cout<<root->val<<' ';
139         inorderTraversal(root->right);
140         return;
141     }
142 };
143 
144 int main() {
145 
146     return 0;
147 }
```

8 Section2

8.1 thm

- 中文測試
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$