


























叽叽喳喳动物园模板

-    叽叽喳喳动物园模板
 - 算法 & 数据结构
 - 最大公约数
 -  辗转相除法 模板
 - 快速幂
 -  快速幂 模板
 - 逆序对
 -  求逆序对数 模板
 - 背包
 -  01背包二维数组 模板
 -  01背包二维数组 模板
 -  01背包滚动数组 模板
 -  01背包滚动数组 模板
 -  完全背包滚动数组 模板
 - 二叉树
 -  前序中序求后序 模板
 - 大数计算
 -  大数加法 模板
 -  大数减法 模板
 -  大数乘法 模板
 -  大数除法 模板
 - 判断素数
 -  判断素数 模板
 - KMP
 -  KMP 模板
 - 并查集
 -  并查集 模板
 -  并查集 模板
 -  并查集 模板
 - DFS
 -  DFS 模板
 - BFS

-  BFS 模板
- STL
 - algorithm
 - merge()
 - next_permutation()
 - sort()
 - queue
 - priority_queue()

算法 & 数据结构

最大公约数

辗转相除法 模板

```
int gcd(int x,int y){  
    if(!y) return x;  
    return gcd(y, x % y);  
}
```

快速幂

快速幂 模板

```
typedef long long ll;  
  
ll qPow(ll x, ll y){  
    ll res = 1;  
    while(y){  
        if(y & 1) res *= x;  
        x *= x;  
        y >>= 1;  
    }  
    return res;  
}
```

逆序对

求逆序对数 模板

```

#include<iostream>
#include<cstdio>

using namespace std;

int n, arrA[500009] = {0}, arrB[500009] = {0};
long long sum = 0;

void solve(int left, int right){
    if(left >= right) return;
    int mid = (left + right) / 2;
    solve(left, mid), solve(mid + 1, right);
    int i = left, j = mid + 1, index = left;
    while(i <= mid && j <= right){
        if(arrA[i] > arrA[j]){// 逆序
            arrB[index++] = arrA[j++];
            sum += mid - i + 1;// i处元素大于j处元素 则i后面的元素也大于j处元素
        }else arrB[index++] = arrA[i++];
    }
    while(i <= mid) arrB[index++] = arrA[i++];
    while(j <= right) arrB[index++] = arrA[j++];
    for(int k = left; k <= right; k++) arrA[k] = arrB[k];
}

int main(){
    // freopen("input.txt", "r", stdin);
    cin>>n;
    for(int i = 1; i <= n; i++){
        scanf("%d", &arrA[i]);
    }
    solve(1, n);
    cout<<sum<<endl;
    return 0;
}

```

背包

👤 01背包二维数组 模板

```

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

using namespace std;

int m, n; // m: 物品总数 n: 背包重量
int dp[1010][1010];

struct node{
    int w;
    int v;
}nn[1010];
//结构用来存放物品重量和价值

void nodee(){
    int i, j;
    memset(dp,0,sizeof(dp));
    for(i = 1; i <= m; i++) {
        for(j = 0; j <= n; j++) {
            if(nn[i].w > j) {
                dp[i][j] = dp[i - 1][j]; // 判断该物品与当前储存背包大小，如果大于，则不取。
            }else {
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - nn[i].w] + nn[i].v); // 反之判断价值
            }
        }
    }
}

int main()
{
    cin>>m>>n;
    for(int i = 1; i <= m; i++) {
        cin>>nn[i].v;
    }
    for(int i = 1; i<= m; i++) {
        cin>>nn[i].w;
    }
    nodee();
    cout<<dp[m][n]<<endl;
    return 0;
}

```

01背包二维数组 模板

```

#include<iostream>
#include<cstring>

using namespace std;

int _size[3500] = {0}; // 占用体积
int _value[3500] = {0}; // 价值
int dp[1009][1009] = {0}; // 结果集

/*
    01背包二维数组
*/

int main(){
    int n, maxSize; // 物体数量和背包容量
    cin>>n>>maxSize;
    for(int i = 1; i <= n; i++){
        cin>>_size[i]>>_value[i];
    }
    for(int i = 1; i <= n; i++){
        for(int j = 0; j <= maxSize; j++){
            if(_size[i] <= j){
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - _size[i]] + _value[i]);
            }else{
                dp[i][j] = dp[i - 1][j];
            }
        }
    }
    cout<<dp[n][maxSize];
    return 0;
}

```

01背包滚动数组 模板

```

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

using namespace std;

int m, n; // m: 物品总数 n: 背包重量
int dp[1010];

struct node{
    int w;
    int v;
}nn[1010];
//结构用来存放物品重量和价值

void nodee(){ // 滚动数组
    int i, j;
    memset(dp,0,sizeof(dp));
    for(i = 1; i <= m; i++) {
        for(j = n; j >= nn[i].w; j--) {
            dp[j] = max(dp[j], dp[j - nn[i].w] + nn[i].v);
        }
    }
}

int main()
{
    cin>>m>>n;
    for(int i = 1; i <= m; i++) {
        cin>>nn[i].v;
    }
    for(int i = 1; i<= m; i++) {
        cin>>nn[i].w;
    }
    nodee();
    cout<<dp[n]<<endl;
    return 0;
}

```

01背包滚动数组 模板

```

#include<iostream>
#include<cstring>

using namespace std;

int _size[3500] = {0}; // 占用体积
int _value[3500] = {0}; // 价值
int dp[13000] = {0}; // 结果集
/*
    01背包滚动数组
*/
int main(){
    int n, maxSize; // 物体数量和背包容量
    cin>>n>>maxSize;
    for(int i = 1; i <= n; i++){
        cin>>_size[i]>>_value[i];
    }
    for(int i = 1; i <= n; i++){
        for(int j = maxSize; j >= _size[i]; j--){
            dp[j] = max(dp[j], dp[j - _size[i]] + _value[i]);
        }
    }
    cout<<dp[maxSize];
    return 0;
}

```

🐼 完全背包滚动数组 模板

```

#include<iostream>
#include<cstring>

using namespace std;

int _size[13000] = {0}; // 占用体积
int _value[13000] = {0}; // 价值
int dp[13000] = {0}; // 结果集

/*
    完全背包滚动数组
*/

int main(){
    int n, maxSize; // 物体数量和背包容量
    cin>>n>>maxSize;
    for(int i = 1; i <= n; i++){
        cin>>_size[i]>>_value[i];
    }
    for(int i = 1; i <= n; i++){
        for(int j = _size[i]; j <= maxSize; j++){
            dp[j] = max(dp[j], dp[j - _size[i]] + _value[i]);
        }
    }
    cout<<[maxSize];
    return 0;
}

```

二叉树

🐼 前序中序求后序 模板


```

#include<iostream>
#include<string>

using namespace std;

string pre, mid;
int n = -1;
/*
    beginIndex从0 , endIndex从长度减一
    注意n需要初始化-1
*/
void solve(int beginIndex, int endIndex){
    if(beginIndex > endIndex) return;
    n++;
    int i;
    for(i = beginIndex; i <= endIndex; i++){
        if(pre[n] == mid[i]) break;
    }
    solve(beginIndex, i - 1);
    solve(i + 1, endIndex);
    cout<<mid[i];
}

int main() {
    while(cin>>pre>>mid) {
        solve(0, pre.length() - 1);
        n = -1;
        cout<<endl;
    }
    return 0;
}

```

大数计算

大数加法 模板

```

#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
#define n 10
/*
    大数加法
    如果遇到加负数的情况需要使用减法模版
*/
string dezero(string a)//用来去掉正数前面的0，也就是说可以输入000001类似这样的数字
{
    long int i;
    for(i=0;i<a.length();i++)
    {
        if(a.at(i)>48) break;
    }
    if(i==a.length()) return "0";
    a.erase(0,i);
    return a;
}
string add(string a,string b)//自然数加法
{
    a=dezero(a);
    b=dezero(b);
    string c;
    string d="0";
    long int kmin,kmax,i;
    reverse(a.begin(),a.end());
    reverse(b.begin(),b.end());
    if(a.length()>b.length()) {kmin=b.length();kmax=a.length();c=a;}
    else {kmin=a.length();kmax=b.length();c=b;}
    c.insert(c.length(),d);
    for(i=0;i<kmin;i++)
    {
        if(a.at(i)>=48&&a.at(i)<=57) a.at(i)--48;
        if(a.at(i)>=97&&a.at(i)<=122) a.at(i)--87;
        if(b.at(i)>=48&&b.at(i)<=57) b.at(i)--48;
        if(b.at(i)>=97&&b.at(i)<=122) b.at(i)--87;
        c.at(i)=a.at(i)+b.at(i);
    }
    for(i=kmin;i<kmax+1;i++)
    {
        if(c.at(i)>=48&&c.at(i)<=57) c.at(i)--48;
        if(c.at(i)>=97&&c.at(i)<=122) c.at(i)--87;
    }
    for(i=0;i<kmax;i++)
    {
        if(c.at(i)>=n)
        {
            c.at(i+1)+=c.at(i)/n;
            c.at(i)=c.at(i)%n;
        }
    }
}

```

```

    }
}
if(c.at(kmax)==0)
{
    c.erase(kmax,kmax+1);
}
for(i=0;i<c.length();i++)
{
    if(c.at(i)>=10) c.at(i)+=87;
    if(c.at(i)<10) c.at(i)+=48;
}
reverse(c.begin(),c.end());
return c;
}
int main()
{
    string a,b;
    while(cin>>a>>b)
    {
        cout<<add(a,b);
    }
    return 0;
}

```

大数减法 模板

```

#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
#define n 10
/*
    大数减法
    同样不支持负数加减
*/
string dezero(string a)//用来去掉正数前面的0，也就是说可以输入000001类似这样的数字
{
    long int i;
    for(i=0;i<a.length();i++)
    {
        if(a.at(i)>'0') break;
    }
    if(i==a.length()) return "0";
    a.erase(0,i);
    return a;
}
int judge(string a,string b)//判断两个正数的大小
{
    if(a.length()>b.length()) return 1;
    if(a.length()<b.length()) return -1;
    long int i;
    for(i=0;i<a.length();i++)
    {
        if(a.at(i)>b.at(i)) return 1;
        if(a.at(i)<b.at(i)) return -1;
    }
    return 0;
}
string sub(string a,string b)//自然数减法
{
    a=dezero(a);
    b=dezero(b);
    long int i,j=0;
    string c="0";
    string c1,c2;
    string d="-";
    if(judge(a,b)==0) return c;
    if(judge(a,b)==1)
    {
        c1=a;
        c2=b;
    }
    if(judge(a,b)==-1)
    {
        c1=b;
        c2=a;
        j=-1;
    }
}

```

```

}
reverse(c1.begin(),c1.end());
reverse(c2.begin(),c2.end());
for(i=0;i<c2.length();i++)
{
    if(c2.at(i)>=48&& c2.at(i)<=57) c2.at(i)-=48;
    if(c2.at(i)>=97&& c2.at(i)<=122) c2.at(i)-=87;
}
for(i=0;i<c1.length();i++)
{
    if(c1.at(i)>=48&& c1.at(i)<=57) c1.at(i)-=48;
    if(c1.at(i)>=97&& c1.at(i)<=122) c1.at(i)-=87;
}
for(i=0;i<c2.length();i++)
{
    c1.at(i)=c1.at(i)-c2.at(i);
}
for(i=0;i<c1.length()-1;i++)
{
    if(c1.at(i)<0)
    {
        c1.at(i)+=n;
        c1.at(i+1)--;
    }
}
for(i=c1.length()-1;i>=0;i--)
{
    if(c1.at(i)>0) break;
}
c1.erase(i+1,c1.length());
for(i=0;i<c1.length();i++)
{
    if(c1.at(i)>=10) c1.at(i)+=87;
    if(c1.at(i)<10) c1.at(i)+=48;
}
reverse(c1.begin(),c1.end());
if(j==-1) c1.insert(0,d);
return c1;
}

int main()
{
    string a,b;
    while(cout<<"input:"&&cin>>a>>b)
    {
        cout<<"output:"<<sub(a,b)<<endl;
    }
    return 0;
}

```

```

#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
#define n 10
/*
    大数乘法
    整数乘法，正整数，负整数，0均可，主要思想就是乘法的笔算方法
*/
string dezero(string a)//用来去掉正数前面的0，也就是说可以输入000001类似这样的数字
{
    long int i;
    for(i=0;i<a.length();i++)
    {
        if(a.at(i)>48) break;
    }
    if(i==a.length()) return "0";
    a.erase(0,i);
    return a;
}
string multiply(string a,string b)//整数
{
    long int i,j,k,yao=0,kai;
    string c1,c2;
    string c3=a+b;
    if(a.at(0)=='-')
    {
        a.erase(0,1);
        yao++;
    }
    if(b.at(0)=='-')
    {
        b.erase(0,1);
        yao++;
    }
    a=dezero(a);
    b=dezero(b);
    if(a.at(0)==48||b.at(0)==48) return "0";
    if(a.length()>b.length())
    {
        c1=a;
        c2=b;
    }
    else
    {
        c1=b;
        c2=a;
    }
    reverse(c1.begin(),c1.end());
    reverse(c2.begin(),c2.end());
    for(i=0;i<c2.length();i++)

```

```

{
    if(c2.at(i)>=48&& c2.at(i)<=57) c2.at(i)-=48;
    if(c2.at(i)>=97&& c2.at(i)<=122) c2.at(i)-=87;
}
for(i=0;i<c1.length();i++)
{
    if(c1.at(i)>=48&& c1.at(i)<=57) c1.at(i)-=48;
    if(c1.at(i)>=97&& c1.at(i)<=122) c1.at(i)-=87;
}
for(i=0;i<c3.length();i++) c3.at(i)=0;
for(i=0;i<c2.length();i++)
{
    for(j=0;j<c1.length();j++)
    {
        kai=c2.at(i)*c1.at(j);
        c3.at(i+j+1)+=kai/n;
        c3.at(i+j)+=kai%n;
        for(k=i+j;k<c3.length()-1;k++)
        {
            if(c3.at(k)>=n)
            {
                c3.at(k+1)+=c3.at(k)/n;
                c3.at(k)=c3.at(k)%n;
            }
            else
            {
                break;
            }
        }
    }
}
for(i=c3.length()-1;i>=0;i--)
{
    if(c3.at(i)>0) break;
}
c3.erase(i+1,c3.length());
for(i=0;i<c3.length();i++)
{
    if(c3.at(i)>=10) c3.at(i)+=87;
    if(c3.at(i)<10) c3.at(i)+=48;
}
reverse(c3.begin(),c3.end());
if(yao==1) c3="-"+c3;
return c3;
}
int main()
{
    string a,b;
    while(cout<<"input:"&&cin>>a>>b)
    {
        cout<<"output:"<<multiply(a,b)<<endl;
    }
}

```

```
}  
return 0;  
}
```

大数除法 模板


```

#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
#define n 10
/*
    大数除法
    用到减法函数sub
    除以0输出error
*/
string dezero(string a)//用来去掉正数前面的0，也就是说可以输入000001类似这样的数字
{
    long int i;
    for(i=0;i<a.length();i++)
    {
        if(a.at(i)>'0') break;
    }
    if(i==a.length()) return "0";
    a.erase(0,i);
    return a;
}
int judge(string a,string b)//判断两个正数的大小
{
    if(a.length()>b.length()) return 1;
    if(a.length()<b.length()) return -1;
    long int i;
    for(i=0;i<a.length();i++)
    {
        if(a.at(i)>b.at(i)) return 1;
        if(a.at(i)<b.at(i)) return -1;
    }
    return 0;
}
string sub(string a,string b)//自然数减法（在之前博客中写到过，这里直接挪过来调用了）
{
    a=dezero(a);
    b=dezero(b);
    long int i,j=0;
    string c="0";
    string c1,c2;
    string d="-";
    if(judge(a,b)==0) return c;
    if(judge(a,b)==1)
    {
        c1=a;
        c2=b;
    }
    if(judge(a,b)==-1)
    {
        c1=b;
        c2=a;
    }
}

```

```

        j=-1;
    }
    reverse(c1.begin(),c1.end());
    reverse(c2.begin(),c2.end());
    for(i=0;i<c2.length();i++)
    {
        if(c2.at(i)>=48&& c2.at(i)<=57) c2.at(i)-=48;
        if(c2.at(i)>=97&& c2.at(i)<=122) c2.at(i)-=87;
    }
    for(i=0;i<c1.length();i++)
    {
        if(c1.at(i)>=48&& c1.at(i)<=57) c1.at(i)-=48;
        if(c1.at(i)>=97&& c1.at(i)<=122) c1.at(i)-=87;
    }
    for(i=0;i<c2.length();i++)
    {
        c1.at(i)=c1.at(i)-c2.at(i);
    }
    for(i=0;i<c1.length()-1;i++)
    {
        if(c1.at(i)<0)
        {
            c1.at(i)+=n;
            c1.at(i+1)--;
        }
    }
    for(i=c1.length()-1;i>=0;i--)
    {
        if(c1.at(i)>0) break;
    }
    c1.erase(i+1,c1.length());
    for(i=0;i<c1.length();i++)
    {
        if(c1.at(i)>=10) c1.at(i)+=87;
        if(c1.at(i)<10) c1.at(i)+=48;
    }
    reverse(c1.begin(),c1.end());
    if(j== -1) c1.insert(0,d);
    return c1;
}

string divide(string a,string b)//自然数除法
{
    if(b.length()==1&&b.at(0)==48) return "error";
    long int i,j;
    string c1,c2,d,e;
    if(judge(a,b)==0) return "1";
    if(judge(a,b)==-1)
    {
        return "0";
    }
    c1=dezero(a);

```

```

c2=dezero(b);
d="";
e="";
for(i=0;i<c1.length();i++)
{
    j=0;
    d=d+c1.at(i);
    d=dezero(d);
    while(judge(d,b)>=0)
    {
        d=sub(d,b);//调用之前的减法函数sub，在本文中也加了进来
        d=dezero(d);
        j++;
    }
    e=e+"0";
    e.at(i)=j;
}
for(i=0;i<e.length();i++)
{
    if(e.at(i)>=10) e.at(i)+=87;
    if(e.at(i)<10) e.at(i)+=48;
}
e=dezero(e);
return e;
}
int main()
{
    string a,b;
    while(cout<<"input:"&&cin>>a>>b)
    {
        cout<<"output:"<<divide(a,b)<<endl;
    }
    return 0;
}

```

判断素数

判断素数 模板

```

bool isPrime(int n) {
    if(n == 2 || n == 3) return true;
    if(n % 6 != 1 && n % 6 != 5) return false;
    // gcc提交 sqrt() 里可以直接放 n，c++ 提交需转化类型即 (double)n，编译失败注意
    for(int i = 5; i <= (int)sqrt(n); i += 6) {
        if (n % i == 0 || n % (i + 2) == 0) return false;
    }
    return true;
}

```

KMP

KMP 模板

```
int _next[10009];
string strA, strB;

void setNext() {
    _next[0] = 0;
    int i = 1, j = 0;
    while(i < strB.length()) {
        if(strB[i] == strB[j]) {
            _next[i++] = ++j;
        } else if(j != 0) {
            j = _next[j - 1];
        } else {
            _next[i++] = 0;
        }
    }
}

int kmp() {
    setNext();
    int i = 0, j = 0;
    while(i < strA.length() && j < strB.length()) {
        if(strA[i] == strB[j]) {
            i++, ++j;
        } else if(j != 0) {
            j = _next[j - 1];
        } else {
            i++;
        }
    }
    if(j == strB.length()) return i - j; // 返回初次匹配坐标
    else return -1; // 不匹配返回-1
}
```

并查集

并查集 模板

```

int arrA[55]= {0};

void init(int x){
    while(x != 0){
        arrA[x] = x;
        x--;
    }
}

int find(int x) {
    if(arrA[x] != x) return find(arrA[x]);
    else return x;
}

void un(int a, int b) {
    int pareA = find(a);
    int pareB = find(b);
    arrA[pareB] = pareA;
}

```

并查集 模板

```

#include<vector>

int re; // 剩余总集合数量

// vectorA要初始化 · re要赋初值

int getRoot(int x, vector<int> &vectorA) {
    if (x != vectorA[x]) {
        vectorA[x] = getRoot(vectorA[x], vectorA);
    }
    return vectorA[x];
}

void merge(int a, int b, vector<int> &vectorA) {
    int aa = getRoot(a, vectorA);
    int bb = getRoot(b, vectorA);
    if (aa != bb) {
        vectorA[bb] = aa;
        re--;
    }
}

```

并查集 模板

- 非递归实现路径压缩

```

int pre[1010]; //存放第i个元素的父节点

int unionsearch(int root) //查找根结点
{
    int son, tmp;
    son = root;
    while(root != pre[root]) //寻找根结点
        root = pre[root];
    while(son != root) //路径压缩
    {
        tmp = pre[son];
        pre[son] = root;
        son = tmp;
    }
    return root;
}

void join(int root1, int root2) //判断是否连通，不连通就合并
{
    int x, y;
    x = unionsearch(root1);
    y = unionsearch(root2);
    if(x != y) //如果不连通，就把它们所在的连通分支合并
        pre[x] = y;
}

```

DFS

DFS 模板

```

void dfs() {
    if (到达终点状态) {
        ...//根据题意添加
        return;
    }
    if (越界或者是不合法状态) {
        return;
    }
    if (特殊状态) {
        // 剪枝
        return;
    }
    for (扩展方式) {
        if (扩展方式所达到的状态合法) {
            修改操作; // 根据题意添加
            标记;
            dfs();
            还原标记;
            // 是否还原标记根据题意
            // 如果加上 ( 还原标记 ) 就是 回溯法
        }
    }
}

```

BFS

BFS 模板

```

void bfs(){
    queue<int> que;
    que.push(1);
    while(que.size()){
        if(到达终点状态){
            //...
            return;
        }
        if(不合法返回){
            return;
        }
        if(特殊状态){
            //时间重置、记录总数、剪枝等等
        }
        for(扩展){
            if(如果能够到达且状态合法){
                // 标记
                queue.push(1);
            }
        }
        que.pop();
    }
}

```

STL

algorithm

merge()

- 语法

```
void merge( list &l1, list &l2 );
```

```
void merge( list &l1, list &l2, Comp compfunction );
```

merge()函数把自己和l2链表连接在一起，产生一个整齐排列的组合链表。如果指定compfunction，则将指定函数作为比较的依据。

- 使用模板


```
// merge algorithm example
#include <iostream>      // std::cout
#include <algorithm>     // std::merge, std::sort
#include <vector>        // std::vector

int main () {
    int first[] = {5,10,15,20,25};
    int second[] = {50,40,30,20,10};
    std::vector<int> v(10);

    std::sort (first,first+5);
    std::sort (second,second+5);
    std::merge (first,first+5,second,second+5,v.begin());

    std::cout << "The resulting vector contains:";
    for (std::vector<int>::iterator it=v.begin(); it!=v.end(); ++it)
        std::cout << ' ' << *it;
    std::cout << '\n';

    return 0;
}

// Output:
// The resulting vector contains: 5 10 10 15 20 20 25 30 40 50
```

next_permutation()

- 语法

```
bool next_permutation (BidirectionalIterator first,BidirectionalIterator last);

bool next_permutation (BidirectionalIterator first,BidirectionalIterator last, Compare comp);
```

`next_permutation()` 会生成一个序列的重排列，它是所有可能的字典序中的下一个排列，默认使用 `<` 运算符来做这些事情。它的参数为定义序的迭代器和一个返回布尔值的函数，这个函数在下一个排列大于上一个排列时返回 `true`，如果上一个排列是序列中最大的，它返回 `false`，所以会生成字典序最小的排列。

- 使用模板

```

int main(){
    int len = 5;
    int b[] = {1,2,3,4,5};
    for(int i=0;i<120;i++){// 120 = 5 * 4 * 3 * 2 * 1
        for(int j=0;j<len;j++){
            cout<<b[j];
        }
        cout<<endl;
        next_permutation(b,b + len);
    }
    return 0;
}

```

sort()

- 语法

```

void sort(Array.begin(), Array.end());
void sort(Array.begin(), Array.end(), cmp);

```

sort()函数为链表排序，默认是升序。如果指定compfunction的话，就采用指定函数来判定两个元素的大小。

- 使用模板

```

typedef struct myNode{
    int score;
    int time;
} node;

bool cmp(node o1, node o2){
    if(o1.score == o2.score){
        return o1.time > o2.time;// 如果o1.score等于o2.score 那么按time从大到小排列
    }else{
        return o1.score > o2.score;// 二者不相等 按score从大到小排列
    }
}

node a[max_size];

sort(a, a + n, cmp);

```

queue

priority_queue()

- 语法

```
priority_queue<Type, Container, Functional>
//升序队列
priority_queue <int, vector<int>, greater<int> > q;
//降序队列
priority_queue <int, vector<int>, less<int> > q;
```

排序算法稳定性：**不稳定**

Type 就是数据类型， Container 就是容器类型（ Container 必须是用数组实现的容器，比如 vector , deque 等等，但不能用 list 。STL 里面默认用的是 vector ）， Functional 就是比较的方式，当需要用自定义的数据类型时才需要传入这三个参数，使用基本数据类型时，只需要传入数据类型，默认是大顶堆，由大到小出队。

- 使用模板

```
// pair 比较，第一元素相同，比较第二元素
priority_queue<pair<int, int> > a;

// 自定义类型
// 方法1
struct tmp1 // 运算符重载 <
{
    int x;
    tmp1(int a) {x = a;}
    bool operator<(const tmp1& a) const
    {
        return x < a.x; // 大顶堆
    }
};

// 方法2
struct tmp2 // 重写仿函数
{
    bool operator() (tmp1 a, tmp1 b)
    {
        return a.x < b.x; // 大顶堆
    }
};
```