一.树状数组

```
int getSum(int x){                  求 c[1,x]的 sum
    int sum = 0;
    for(int i = x;i >= 1;i -=lowbit(i))
        sum+=c[i];
    return sum;
}

void update(int x,int v){
    for(int i = x;i < MAXN;i+=lowbit(i))
        c[i]+=v;
}
```

二.DP

1. 最大连续子序列：dp[i] = max(A[i],dp[i]+A[i]);
2. 最长不下降子序列：dp[i] = max(1,dp[j]+1) (j=1,2,……,i-1, A[j]<=A[i])
3. 最长回文串:

```
    for(int i = 0;i < n;i++){
        dp[i][i] = 1;
        if(i>0&&A[i-1]==A[i]){
            dp[i-1][i] = 1;
            ans = 2;
        }
    }
    for(int L = 3;L <= len;L++){
        for(int i = 0;i + len - 1 < len;i++){
            int j = i+len-1;
            if(dp[i+1][j-1]==1&&A[i]==A[j]){
                dp[i][j]=1;ans = L;
            }
        }
    }
```

4. 最长公共子序列:

```
    dp[i][j] = {
        dp[i-1][j-1]+1; A[i]=B[j];
        max(dp[i-1][j],dp[i][j-1]);
    }
```

5. DAG

6. 背包问题:

```
void beibao(){
    for(int i = 1;i <= n;i++){
```

```
            for(int v = V;v >= w[i];v--){            这是 01 背包问题
            for(int v = w[i];v<=V;v++){              这是完全背包问题
                dp[v] = max(dp[v],dp[v-w[i]]+c[i]);
            }
        }
}
```

三.图

1. 最短路径

(1).Dijkstra

```
 void Dijkstra(int s){
        fill(dis,dis+MAXN,inf);
        fill(vis,vis+MAXN,false);
        dis[s] = 0;
        for(int i = 0;i < n;i++){
            int u =-1,MIN = inf;
            for(int j= 0;j < n;j++){
                if(!vis[j]&&dis[j]<MIN){
                    u = j,MIN = dis[j];
                }
            }
            if(u==-1)return;vis[u]=true;
            for(int v = 0;v < n;v++){
                if(!vis[[v]&&G[u][v]!=inf){
                    if(G[u][v]+dis[u] < dis[v]){

                    }else if(G[u][v]+dis[u]==dis[v] &&  其他条件){

                    }
                }
            }
        }
 }
 void getPath(int s){
        if(s==start){
            tempPath.push_back(s);
            1.在这里计算对应的值
            2.计算完后进行相应的更新
            3.达到条件时  path = tempPath;
            tempPath.pop_back();
        }
        tempPath.push_back(s);
        for(int i = 0;i < pre[s].size();i++){
            getPath(pre[s][i]);
        }
```

```
        tempPath.pop_back();
    }
```

(2).SPFA

```
    bool SPFA(int s){
        memset(inq,false,sizeof(inq));    inq 记录顶点是否在队列中
        memset(num,0,sizeof(num));          num 数组记录顶点的入队次数
        fill(d,d+MAXN,INF);
        queue<int> q;
        q.push(s);
        while(!q.empty()){
            int u = q.front();q.pop();
            inq[u] = false;                      u 不在队列中
            for(int j = 0;j < Adj[u].size();j++){
                int v = Adj[u][j].v;
                int dis = Adj[u][j].dis;
                if(d[u]+dis < d[v]){
                    d[v] = d[u]+dis;
                    if(!inq[v]){
                        q.push(v);
                        inq[v] = true;
                        num[v]++;
                        if(num[v]>=n)return false;
                    }
                }
            }
        }
        return true;
    }
```

2. 最小生成树(Prim 算法)

```
int prim(){
    fill(d,d+MAXN,inf);
    d[0] = 0;
    int ans = 0;
    for(int i = 0;i < n;i++){
        int u = -1,MIN = inf;
        for(int j = 0;j < n;j++){
            if(vis[j]==false && d[j]<MIN){
                u = j;MIN = d[j];
            }
        }
        if(u==-1)return false;
        vis[u]=true;
        ans += d[u];
```

```
            for(int v= 0;v < n;v++){
                if(!vis[v]&&G[u][v]!=inf){
                    if(G[u][v] < d[v]){
                        d[v] = G[u][v];
                    }
                }
            }
        }
        return ans;
}
```

3. 图的遍历

(1). DFS

```
void DFS(int s){
        vis[s] = true;
        这里是一些操作,包括剪枝之类的
        for(int v = 0;v < n;v++){
            if(!vis[v]&&G[s][v]!=inf){
                DFS(v);
            }
        }
}
void DFSTravel(){
        for(int i = 0;i < n;i++){
            if(!vis[i])DFS(i);
        }
}
```

(2).BFS

```
void BFS(int s){
        queue<int> q;
        q.push(s);inq[s]=true;
        while(!q.empty()){
            int u = q.front();q.pop();
            for(int v = 0; v< n;v++){
                //计层，用结构体的话，放在这里
                if(!vis[v]&&G[u][v]!=inf){
                    q.push(v);
                    inq[v] = true;
                    //计层的话,开数组放在这里，否则会被覆盖
                }
            }
        }
}
```

```
void BFSTravel(){
    for(int i = 0;i < n;i++){
        if(!inq[i])BFS(i);
    }
}
```

4.图论的题
这种题只能根据题目要求进行判断，另外判断成环，可以用 set 元素个数和读入结点个数是否相等来判断

5.拓扑排序

四.栈,队列,哈希,链表
1.链表：数据与 next 分离，剔除无用结点，长度为 0 单独考虑

2.栈：如果考一个序列的所有出栈情况,最笨的方法是排列后分别判断;
判断一个序列是否是出栈顺序:

```
bool is(){
    for(int i = 1;i <= n;i++){
        s.push(i);
        if(q.front()==i){
            while(!s.empty()&&q.front()==s.top()){
                q.pop();s.pop();
            }
        }
        if(!s.empty())return false;
        else return true;
    }
}
```

3.队列:容易题不常见,难的就是银行排队的题目,见机行事吧
4.hash(二次探测法)

```
bool insert(int x){              二次探测法插入
    for(int i = 0;i < size;i++){
        int index = (x+i*i)%size;     这里有的把 i*i 放在外面,根据题意判断
        if(hash[index]==-1){
            hash[index] = x;
            return true;
        }
    }
    return false;
}

int search(int x){
    int cnt = 0;
```

```
    for(int i = 0;i <= size;i++){     与插入不同,这里带等号
        int index = (x+i*i)%size;
        if(hash[index]!=x && hash[index]!=-1){
            cnt++:
        }
    }
    return cnt;
}
```

五. 树

1. 红黑树

判断方法：根为黑色，所有红结点的儿子都是黑节点，所有结点左右子树的(黑节点)高度相等

```
bool judge1(int s){              判断所有红节点的儿子都是黑节点
    if(s == -1)return true;
    if(red[s]){
        if(T[s].l!=-1&&red[T[s].l])return false;
        if(T[s].r!=-1&&red[T[s.r]])return false;
    }
    return judge1(T[s].l)&&judge1(T[s].r);
}

bool judge2(int s){              判断所有结点左右子树的(黑结点)高度相等
    if(s == -1)return false;
    int L = getH(T[s].l);
    int R = getH(T[s].r);
    if(L!=R)return false;
    return judge2(T[s].l)&&judge2(T[s].r);
}

int getH(int s){
    if(s == -1)return 0;
    int L = getH(T[s].l);
    int R = getH(T[s].r);
    return max(L,R)+1;          这是正常求树高
    return black[s]?max(L,R)+1:max(L,R);      这是求(黑结点)高度
}
```

2.平衡二叉树

四种旋法：

(1).插在左子树的左子树:树右旋

(2).插在左子树的右子树:左子树左旋，然后树右旋

(3).插在右子树的右子树:树左旋

(4).插在右子树的左子树:右子树右旋，然后树左旋

```
int leftRotate(int s){                    左旋
    int temp = T[s].r;
    T[s].r = T[temp].l;
    T[temp].l = s;
    return temp;
}
int rightRotate(int s){                   右旋
    int temp    = T[s].l;
    T[s].l = T[temp].r;
    T[temp}.r = s;
    return temp;
}
int leftRightRoate(int s){                左右旋
    T[s].l = leftRotate(T[s].l);
    return rightRotate(s);
}
int rightLeftRotate(int s){               右左旋
    T[s].r = rightRotate(T[s].r);
    return leftRotate(s);
}

int insertT(int& s,int k){
    if(s == -1) return newnode(k);
    else if(k < T[s].k){
        T[s].l = insertT(T[s].l,k);
        int L = getH(T[s].l),R = getH(T[s].r);
        if(L - R > 1){
            if(k < T[T[s].l].k){
                s = rightRotate(s);
            }else{
                s = leftRightRotate(s);
            }
        }
    }else{
        T[s].r = insertT(T[s].r,k);
        int L = getH(T[s].l),R = getH(T[s].r);
        if(R-L > 1){
            if(k > T[T[s].r].k)
                s = leftRotate(s);
            else
                s = rightLeftRotate(s);
        }

    }
```

```
    return s;
}
```

3.堆 (堆首先必须是一个完全二叉树)

(1).判断是大根堆还是小根堆

```
void judge(){
    for(int i = 2;i <= n;i++){
        if(T[i/2]>T[i])small = false;
        if(T[i/2]<T[i])big = false;
    }
    if(!small&&!big)不是堆
    else if(small) 小根堆
    else if(big) 大根堆
}
```

(2)堆排序: 堆排序的特点是后面有 k 个数是排好序的且,这 k 个数都 >= 第一个数

每次从后向前找到第一个 >= T[1]的数 T[p], 交换 T[p]和 T[1]后执行下面的调整算法

```
void adjust(int low,int high){
    int i = low,j = 2*i;
    while(i <= high){
        if(j+1<=high && T[j+1]>T[j])j+=1;
        if(T[i]>=T[j])break;
        swap(T[i],T[j]);
        i = j,j = 2*i;
    }
}
```

4. 完全二叉树(判断是否是完全二叉树或者建树)

(1).建树: 直接用数组模拟

(2).判断

```
void bfs(int s){                    用 bfs 判断否是完全二叉树,如果遇到了非内结点后又遇到内结点则不是
    queue<int> q;
    q.push(s);
    int flag = 0;
    bool isCom = true;;
    while(!q.empty()){
        int u = q.front();q.pop();
        if(T[u].l!=-1){
            if(flag==1) isCom = false;
            q.push(T[u].l);
        }else flag = 1;                遇到叶子结点
        if(T[u].r!=-1){
            if(flag==1) isCom = false;
            q.push(T[u].r);
        }else flag = 1;
```

```
    }
}

bool check(){                                   直接检测
    int half = len/2;
    for(int i= 1;i < half;i++){         所有内结点
        if(T[i].l==-1)return false;
        if(T[i].r==-1)return false;
    }
    for(int i = half+1;i <= n;i++){  所有叶子结点
        if(T[i].l!=-1)return false;
        if(T[i].r!=-1)return false;
    }
    if(T[half].l==-1&&T[half].r!=-1)return false;
    return true;
}
```

5.哈夫曼树

(1).如果不用建树,就用优先队列:priority_queue

(2).如果需要建树,是从下向上建,就要在结点中添加一个指向父节点的值

6.普通二叉树

(1).构建

先序+中序:

后序+中序:

先序+后序: 如果有只有一个儿子的结点, 则会有歧义

中序+层序: 这个暂时还未考

(2).LCA(与上面构建类似)

五种情况: u 是根

v 是根

u,v 在根的左侧

u,v 在根的右侧

u,v 在根的两侧

(3).遍历,先序遍历,后序遍历,中序遍历

```
void BFS(int s){
    queue<int> q;
    q.push(s);
    while(!q.empty()){
        int u = q.front();q.pop();
        for(int i = 0;i < T[u].size();i++){
            int v = T[u][i];
            q.push(v);
        }
    }
```

```
}
void DFS(int s){
	if(s == -1)return;
	for(int i = 0;i < T[s].size();i++){
		int v = T[s][i];
		DFS(v);
	}
}
```

6.并查集：一定要记得初始化

```
int findFa(int x){
	if(x == father[x])return x;
	else {
		int Fa = findFa(father[x]);
		father[x] = Fa;
		return Fa;
	}
}

int union(int x,int y){
	int Fx = findFa(x);
	int Fy = findFa(y);
	father[Fx] = Fy;
	return Fy;
}
```

对于并查集的题目，在统计时要使用 findFa(x)来查找 father，因为 father[x]里面存的可能不是祖先而是直系父亲

7.排序:

1. 快速排序：特点：每次排完,排好的元素在最终结果对应的位置

```
int quick(int low,int high){
	int temp = arr[low];//基准数据
	while(low < high){
		//当队尾元素大于等于基准元素,向前挪动 high 指针
		while(low < high && arr[high] >= temp)
			high--;
		arr[low] = arr[high];
		//当队首指针小于等于基准元素,向后挪动 low 指针
		while(low < high && arr[low] <= temp)
			low++;
		arr[high] = arr[low];
	}
	//将 temp 放在该在的位置
```

```
    arr[low] = temp;
    return low;
}
```
2. 堆排序：前面说了
3. 插入排序：开始 k 个元素排好,后面的与输入序列一样

8.其他

(1).数字转 string

```
    sprintf(chs,"%d",sum,',');//先将数字存在 char 数组中
    string str = chs;                    //将 char 数组转为 string
```

(2).对时间进行排序,可以使用 std 的字符串直接对字符串的时间进行比较。

(3).对于中序，后序先序等互相获得的题目,递归条件可以改成通用

```
    if(preL>preR)return;
    else if(preL == preR){
        post.push_back(pre[preL]);
        return;
    }
```

(4). 对于 AVL 树,注意：左旋右旋一定要写对，如果结果不对很可能是因为左旋右旋插入函数中,插入结点后对树进行调整时 要记得将旋后的返回值赋值给 s,如：s = rightRotate(s)

(5). 除了只涉及完全二叉树和图，一定要建结点，不要浪费时间，有时候用到子树 T[T[s].l]一定要先检查是否为-1

(6). 对于数 1-n 中有多少个 num(1<= num <=9)的题目，对第 i 位进行分离即可,即 ai = num,ai<num,ai>num;分别得出左右数的大小，然后乘在一起等操作即可。

(7). 有些数据可能是 float,结果当 int 处理

(8). 取整:四舍五入:round 向上取整:ceil 向下取整:floor,强转

(9). 浮点转整形时，计算 sum 时，将 sum 全计算完再做，若每次都转再相加，精度严重丢失

(10). hash_set 用法：加上下面两句 #include <hash_set> using namespace __gnu_cxx;

(11). string.rbegin(),rend()是从右向左遍历

(12). atoi(str.c_str())需要用到 stdlib.h 库, cctype 库里有一些用到的函数

(13). cin >> int ;后面跟 getline，getline 会得到一个 \n

最重要两点：

1. 如果暴力能过部分(就是枚举),在找不到好方法的前提下，就用暴力方法
2. 如果用一个临时变量不好做，那就用多个，不要吝啬变量的应用,比如那道统计单词的题目