2019洛谷夏令营普及组NOTE

-by Kuroko

Part Ⅰ：基础数据结构

一：栈——先进后出

PS:C++的STL的内置stack在进行大数据操作时会十分慢，而且可用的操作有限，在处理OJ问题时，可使用普通数组来模拟栈。

STL实现：

# include <stack>

stack <typename> name;

手动实现：动态长度数组（用参数记录尾部位置）

#include <cstdio>

int s[10005];

int tot=0;

// s[1],s[2],s[3],... s[tot]

// ->

// s[1],s[2],s[3],... s[], s[tot](x)

void push(int x)

{

s[++tot]=x;

}

void pop()

{

tot--;

}

int top()

{

return s[tot];

}

void print()

{

for(int i=1;i<=tot;i++)

printf("%d ",s[i]);

puts("");

}

int main(void)

{

push(1);print();printf("Top = %d\n",top());

push(2);print();printf("Top = %d\n",top());

push(3);print();printf("Top = %d\n",top());

pop();print();printf("Top = %d\n",top());

pop();print();printf("Top = %d\n",top());

push(4);print();printf("Top = %d\n",top());

push(5);print();printf("Top = %d\n",top());

pop();print();printf("Top = %d\n",top());

pop();print();printf("Top = %d\n",top());

pop();print();printf("Top = %d\n",top());

return 0;

}

实例：洗碗问题，括号匹配问题，后缀表达式，波兰表达式

括号匹配：给定一个字符串，这个字符串由()[]{}这六个字符构成。如果所有的括弧都可以匹配上，那么就说这个字符串合法，否则非法。下面给一些例子：

* [({})]：合法。
* [([]){}[]]{}：合法。
* {{}：非法。
* ([)(])：非法。

编写一个程序，判断给定字符串是否合法。

#include <cstdio>

#include <stack>

using namespace std;

int main(void)

{

char a[1005];

stack <char> s;

scanf("%s",a);

for(int i=0;a[i];i++)

{

if(a[i] == '(' || a[i] == '[' || a[i] == '{')

s.push(a[i]);

else

{

if(s.empty()) goto GG;

// 如果栈是空的，s.empty()就返回1

if( (s.top() == '(' && a[i] == ')') ||

(s.top() == '[' && a[i] == ']') ||

(s.top() == '{' && a[i] == '}'))

s.pop();

else

goto GG;

}

}

if(!s.empty()) goto GG;

// if(s.size())

puts("Yes");

return 0;

GG:

puts("NO!!!!!!");

return 0;

}

二：队列——先进先出

STL实现：

# include <queue>

queue <typename> name;

手动实现：用动态参数记录头与尾的索引，实现方法同“stack”

循环队列：

const int MAXN = 1<<20;//即 1\*pow(2,20)

struct queue

{

int l,r,q[MAXN];

queue():l(1),r(0){};

inline void push(int x)

{

q[++r & MAXN-1] = x; //正常的手写队列都是使用"%size"的方法

//采用 "&size" 可以使得循环队列的操作快很多，因为大数取模慢于“与”运算

//有一点要注意的是这种操作方法下的 MAXN必须是2的幂次方

}

inline void pop()

{

l++;

}

inline int front()

{

return q[l&MAXN-1];

}

inline bool size()

{

return l <= r;

}

}

实例：约瑟夫问题，时间复杂度O（nk）

三：双端队列

STL实现：

# include <deque>

deque <typename> name;

可以当做链表来用，实现前段和后端的压入&弹出。

#include <deque>

// deque: 双端队列

deque <int> d;

d.push\_front()

d.push\_back()

d.pop\_front()

d.pop\_back()

d.front()

d.back()

d[i] : // d现在的第i个元素 复杂度极高

deque允许通过下标访问元素，但是效率极低，时间复杂度为O(n)。

四：链表——高效插入与删除

STL实现:

# include <list>

list<typename> name;

手动实现:

struct node

{

typename val;

node \*next;

//node \*pre;

};

链表访问和遍历元素的效率很低，但是链表插入与删除元素远优于传统连续内存的数组。

五：优先队列

STL实现：

priority\_queue<Type, Container, Functional>

priority\_queue<int,vector<int>, less<int> > a;

自定义优先规则：重载“<”运算符

struct info

{

int val;

bool operator <(const info &a) const{return a.val < val};

};

priority\_queue<info> a;

//自定义优先规则

e.g

#include <cstdio>

#include <iostream>

#include <queue>

#include <string>

using namespace std;

struct event

{

int t;

string s;

friend bool operator < (event a,event b)

{

return a.t>b.t;

}

event(int \_t=0,string \_s="qwq")

{

t=\_t;

s=\_s;

}

};

priority\_queue <event> q;

int main(void)

{

q.push(event(100," da lai lai"));

q.push(event(50," akioi"));

q.push(event(120," coding"));

cout<< q.top().t << q.top().s << endl;

q.pop();

cout<< q.top().t << q.top().s << endl;

q.pop();

cout<< q.top().t << q.top().s << endl;

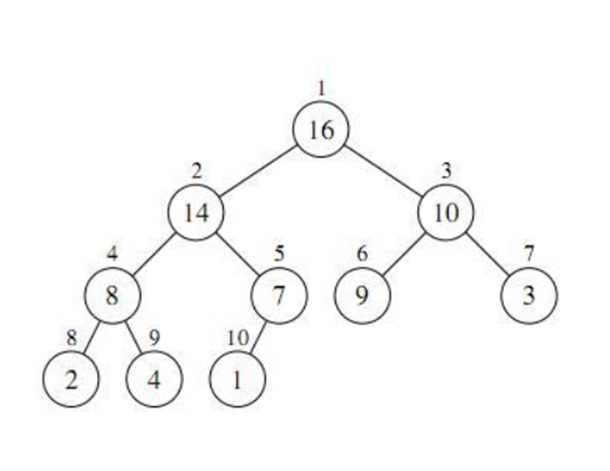
q.pop();

}

优先队列是一个可以在插入新元素时，高效维护有序性的序列。

六：堆

堆是树形结构，每个节点拥有一个key

父亲节点的key必大于两个儿子节点

堆式存储：树的高度尽可能低

堆只能查询根节点，要应对查询操作，只需要返回根节点的key.

给定一个key值，如何将它插入堆？

* 核心思想：修复
* 直接将它摆在堆的末尾，然后修复这个堆
* 如何修复？
* 如果它的key值比父亲大，则交换它和父亲，递归执行。
* 直到它的key值比父亲小，修复完毕。

堆的删除仅限于删除根节点。

如何删掉一个根节点呢？

* 核心思想：修复
* 直接把它和末尾的元素交换，然后直接删掉它
* 现在，原先在末尾的元素到了根节点。如何修复这个堆？
* 现在面对的问题是：可能父节点x比子节点要小。
* 修复方法：在子节点中选个较大的，与x交换。递归执行。

堆排序：

#include <cstdio>

#include <algorithm>

using namespace std;

#define DAD(x) ((x)/2)

#define LE(x) ((x)<<1)

#define RT(x) (((x)<<1)|1)

struct heap

{

int s[500005];

int tot;

int top()

{

return s[1];

}

void modify(int x)

{

if(x==1) return;

if(s[x] > s[DAD(x)])

swap(s[x],s[DAD(x)]),modify(DAD(x));

}

void ins(int key)

{

s[++tot]=key;

modify(tot);

}

void repair(int x)

{

int tar=s[LE(x)]>s[RT(x)]?LE(x):RT(x);

if(s[x]<s[tar])

swap(s[x],s[tar]),repair(tar);

}

void pop()

{

s[1]=s[tot];

s[tot--]=0;

repair(1);

}

};

heap h;

int ans[100005];

int main(void)

{

int n,x;

scanf("%d",&n);

for(int i=1;i<=n;i++)

{

scanf("%d",&x);

h.ins(x);

}

for(int i=1;i<=n;i++)

{

ans[i]=h.top();

h.pop();

}

for(int i=n;i>=1;i--)

printf("%d ",ans[i]);

return 0;

}

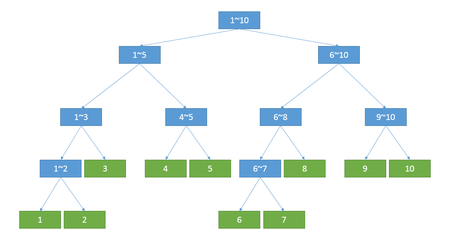
七：线段树

线段树是一种二叉搜索树，与区间树相似，它将一个区间划分成一些单元区间，每个单元区间对应线段树中的一个叶结点。

对于线段树中的每一个非叶子节点[a,b]，它的左儿子表示的区间为[a,(a+b)/2]，右儿子表示的区间为[(a+b)/2+1,b]。

因此线段树是平衡二叉树，最后的子节点数目为N，即整个线段区间的长度。

使用线段树可以快速的查找某一个节点在若干条线段中出现的次数，时间复杂度为O(logN）。而未优化的空间复杂度为2N，因此有时需要离散化让空间压缩。



八：二叉树的三种遍历

二叉树的时候由于要保证有序，同时子节点个数是限定的，直接保存每个节点的左右儿子是谁即可。

* 先序遍历：根->左->右
* 中序遍历：左->根->右
* 后序遍历：左->右->根

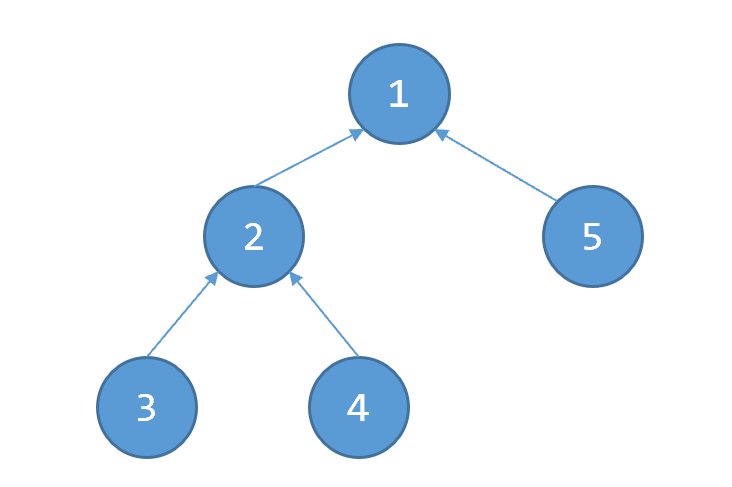
如果一棵树的中序遍历会保证有序，那么我们称这种树为二叉排序树。

换而言之，二叉排序树满足左<根<右这一性质。

PART Ⅱ: ALGORITHM

一：并查集

简要思想：线性相连，同一条线上为统一集合，合并集合时，只需要相连两条线。

实例：亲戚问题（若A是B的朋友，也是C的朋友，则B与C也是朋友等类似问题）

#include <cstdio>

struct unionFind

{

int bin[10005];

unionFind()

{

for(int i=1;i<=10002;i++)

bin[i]=i;

}

int anc(int x)

{

if(x == bin[x]) return x;

return bin[x] = anc(bin[x]);

}

void uni(int x,int y)

{

bin[anc(x)]=anc(y);

}

void ask(int x,int y)

{

printf("%s\n",anc(x)==anc(y)?"Y":"N");

}

};

unionFind u;

int main(void)

{

int n,m,z,x,y;

scanf("%d%d",&n,&m);

while(m--)

{

scanf("%d%d%d",&z,&x,&y);

if(z==1) u.uni(x,y);

else u.ask(x,y);

}

return 0;

}

二：哈希表

简要思想：当n的大小远超过数组可容纳的长度时，取模一个数组长度以内的一个数来存储。

#include <cstdio>

#include <vector>

#include <unordered\_set>

using namespace std;

// 128MB -> 32000000 个int

struct brute

{

bool w[1000005];

void ins(int x)

{

w[x]=1;

}

void ask(int x)

{

printf("%d\n",w[x]);

}

};

#define ha 233333

struct naive

{

bool w[ha + 5];

void ins(int x)

{

w[x % ha]=1;

}

void ask(int x)

{

printf("%d\n",w[x%ha]);

}

};

struct zip

{

vector <int> w[ha + 5];

void ins(int x)

{

w[x%ha].push\_back(x);

}

void ask(int x)

{

for(auto i=w[x%ha].begin();i!=w[x%ha].end();i++)

if(\*i == x) goto done;

puts("No");

return;

done:

puts("Yes");

}

};

zip B;

unordered\_set <int> S;

int main(void)

{

printf("%d\n",S.count(5));

S.insert(5);

printf("%d\n",S.count(5));

return 0;

}

缺点：hash碰撞，例如 5%1234 = 1239%1234,此时结果就会出现错误。

理论期望：当对p取余时，每根号p个里面，就会有两个人碰撞，所以p应取得尽量大。

优化：

1. ：对两个数（甚至三个）取模，当这两个数是质数时，a\*b以内的任意数哈希时一定是有效的。此方法没有改变碰撞几率，而是扩大有效范围。
2. ：拉链法

为每一个取模的结果开一个数组来存储。vector<int> a[n];

虽增大了一定的空间与时间复杂度，但完全保证了正确性。

三：二分法

用于解决有序序列的查找和搜索问题。

#include <cstdio>

int f(int n)

{

if(n>100) return 1;

return 0;

}// 需要找到n=101，它是第一个1

void search()

{

int l=1,r=1024,mid;

while(r - l >=5)

{

printf("[%d, %d]\n",l,r);

mid = (l+r)/2;

if(f(mid) == 0) l=mid;

else r=mid;

}

for(;f(l)==0;l++);

printf("%d\n",l);

}

int main(void)

{

search();

return 0;

}

二分法的思想很简单，可与贪心，前缀和等方法相结合广泛应用于模拟问题中。



四：三分法

二分法用于求单调函数的零点，三分法则用于求单调函数的峰值

将范围区间不断三等分，通过比较中间两端点的函数值来排除一个端区间，不断缩小区间长度最终确定峰值

·————·————·————·

a b c d

若f(b) < f(c),则峰值必不在(a,b);

若f(b) > f(c),则峰值必不在(c,d).

五：排序不等式

排序不等式： 正序和 ≥ 乱序和 ≥ 逆序和

要你钦定一个长度为的排列使得

最小/最大。

最大：a=1,2,3...

最小：a=n,n-1,...

如何证明？ [调整法]

假设某个乱序方案得到了最大值。去证“可以通过交换两个数，来得到更大的值”，从而推翻假设，得证原命题。

六：分块

分块是将整个区间分成若干块，维护整块的信息，以加速查询。

分块维护区间和、积、max/min等

需要实现：快速查找一个点所在的块、快速查找一个块的起点和终点。

分块策略：设块大小为，则在这个块。

最佳分块区间大小：根号N

# include <iostream>

# include <cmath>

# include <cstdio>

using namespace std;

int main(void)

{

int n,m,f,x,y,k,\*a,w,cnt;

long long ans,\*sum;

cin>>n>>m;

a = new int [n];

for(int i = 0;i < n;i++) scanf("%d",&a[i]);

w = int(sqrt(n));

cnt = n%w==0?n/w:n/w+1;

sum = new long long [cnt];

for(int i = 0;i < cnt;i++) sum[i] = 0;

for(int i = 0;i < n;i++) sum[i/w] += a[i];

while(m--)

{

scanf("%d %d %d",&f,&x,&y);

if(f == 1)

{

scanf("%d",&k);

for(int i = x-1;i < y;i++)

{

a[i] += k;

sum[i/w] += k;

}

}

else

{

ans = 0; x--;

while(x < y)

{

if(x%w == 0)

{

if(x+w <= y)

{

ans += sum[x/w];

x += w;

}

else

{

for(int i = x;i < y;i++) ans += a[i];

x = y;

}

}

else

{

for(int i = x;i < x+w-x%w;i++) ans += a[i];

x = x+w-x%w;

}

}

printf("%d\n",ans);

}

}

return 0;

}

七：单调栈

目的：求出最近的满足条件元素

1：按照排序规则清空不满足条件的元素

2：栈顶=答案

3：压入当前元素

4：循环

八：单调队列

目的：在滑动的动态窗口中，求出满足条件的极端元素

pre：先创建一个空的静态窗口

1：弹出所有排序规则下小于当前元素的元素

2：抽象滑动窗口，弹出超出范围元素

3：加入新元素

4：队首 = 答案

九：递推

* 递推：相邻两项或多项数字（或状态）之间存在某种关系，可以通过前一项或多项按照某一规律推出其后一项数字（或状态），或者是通过后一项或多项按照某一规律推出其前一项数字（或状态），即Fn=g(Fn-1)或者Fn-1=g＇(Fn)

典例：斐波那契数列，直线切割

走网格图的方案数量：a[i][j] = a[i][j-1] + a[i-1][j];

十：卡特兰数

1：基本公式形式

1. ：h(0) = h(1) = 1;

h(n)= h(0)\*h(n-1)+h(1)\*h(n-2) + ... + h(n-1)\*h(0) (n>=2)

1. ：h(n)=h(n-1)\*(4\*n-2)/(n+1)

十一：分治

定义：分治就是分而治之，把一个问题划分成若干个同样的小问题，然后各个击破

实例：快速幂，归并排序，求逆序对

PART Ⅲ : TIPS

一：如何撰写博客

1：写博客的用途

算法笔记，题解，大赛游记等。

2：开通博客

选择先有的博客网站/利用WordPress等软件自行搭建。

3：Markdown编辑器

作业部落等……

4：Markdwon语法

* # \* n ：表示n级标题
* \* \*：斜体
* \*\* \*\*：粗体
* \*\*\* \*\*\*：斜粗体
* --- / ------：分割线
* > ：引用
* >\*：加上·符号
* - ：无序表
* 1.，2.……：有序表
* ![描述](链接)：插入图片（用<https://sm.ms>可找出本地图片的链接）
* [描述]（链接）：插入链接
* $ $：数学公式（一小部分）
* $$ $$：数学公式（整行）