程序设计思维与实践

Thinking and Practice in Programming

C++ 与 STL（下）、搜索（上） | 内容负责：张腾/杨家齐

补充

- 关于读入
  - 我们所见的三种形式
  - EOF，为End Of File的缩写，通常在文本的最后存在此字符表示资料结束。

Input

输入第一行是一个整数N，代表接下来N行会有N组样例输入。

Input

题目包含多组数据，每组数据一行. 每行两个整数 a 和 b (1 <= a, b) , 用一个空格分开，某行 a = 0, b = 0, 意味着数据EOF.

Input

输入多组数据，以EOF作为数据的结束
每组数据在一行中包含了两个十六进制整数a和b。

补充

- 对于前两种方式很简单，最后一种的读入的处理方法
- 可用文件测试（课上演示）

Select Code

```
#include<iostream>
using namespace std;
int main(int argc, char** argv)
{
    double a;
    while (cin >> a, a != EOF)
    {
        if (a > 0) cout << a << endl;
        if (a < 0) cout << -a << endl;
    }
}
```

```
while(1)
{
    double a;
    cin>>a;
}
```

Select Code

```
#include<cstdio>
int main() {
    double a=0;
    while(~scanf("%lf",&a)){
        if(a>0)printf("%.2lf\n",a);
        else printf("%.2lf\n",-a);
    }
    return 0;
}
```

Select Code

```
#include <iostream>
#include<cstdio>
using namespace std;
int main()
{
    double a;
    while(cin>>a)
    {
    }
    return 0;
}
```

√

√

补充

- Week 1 实验课 CodeReview 发现的问题
  - W 题，统计每个分数的学生的数目
  - 数组的妙用 —— Counter
  - 静态声明大数组，且 size 不要吝啬，开大亿点点
  - 理解数组声明在栈区/堆区/静态区的时耗不同

#24322072 |

Status	Time	Memory	Length	La
Accepted	109ms	1748kB	271	C-

Select Code

```
#include<stdio.h>

int main()
{
    int a[1000],n,val;
    int sum=0;
    while(scanf("%d",&n)&&n)
    {
        for(int i=0;i<n;i++)
        {
            scanf("%d",&a[i]);
        }
        scanf("%d",&val);
        for(int i=0;i<n;i++)
        {
            if(a[i]==val)
            {
                sum++;
```

#24350761 |

Status	Time	Memory	Length	Lang	
Accepted	124ms	3404kB	368	G++	20

Select Code

```
#include <iostream>
#include<cmath>
using namespace std;

int main() {
    int n;
    while(scanf("%d",&n)&&n!=0)
    {
        getchar();
        int *a=new int[n];
```

```
#include <iostream>
#include <cstring>
#include <memory>
#include <string>
#include <cstdio>

int count_score[101];

int main() {

    while (true) {
```

√

补充

- Week 1 实验课 CodeReview 发现的问题
  - X 题，英文 A+B
  - 常量数组/map的使用

```
int getnumber(string x, int location)//location是
{
    int temp = 0;
    if (x[location] == 'z')temp = 0;
    if (x[location] == 'o')temp = 1;
    if (x[location] == 't')
    {
        if (x[1+ location] == 'w')te
        else temp = 3;
    }
    if (x[location] == 'f')
    {
        if (x[1+ location] == 'o')te
        else temp = 5;
    }
    if (x[location] == 's')
    {
        if (x[1 + location] == 'i')t
        else temp = 7;
    }
    if (x[location] == 'e')temp = 8;
    if (x[location] == 'n')temp = 9;
    return temp;
```

```
int chuli(string s){
    if(s[0]=='=')
        return -1;
    if(s[0]=='+')
        return -2;
    switch(s[0]){
        case 'z':
            return 0;
        case 'o':
            return 1;
        case 't':
            if(s[1]=='w')   return 2;
            if(s[1]=='h')   return 3;
        case 'f':
            if(s[1]=='o')   return 4;
            if(s[1]=='i')   return 5;
        case 's':
            if(s[1]=='i')   return 6;
            if(s[1]=='e')   return 7;
        case 'e':
            return 8;
        case 'n':
            return 9;
    }
}
```

```
int change(string s)
{
    if(s=="zero")return 0;
    if(s=="one")return 1;
    if(s=="two")return 2;
    if(s=="three")return 3;
    if(s=="four")return 4;
    if(s=="five")return 5;
    if(s=="six")return 6;
    if(s=="seven")return 7;
    if(s=="eight")return 8;
    if(s=="nine")return 9;
```

√

```
char a[10][10] = { "zero", "one", "two", "three", "
int change(string str)
{
    for (int i = 0; i < 10; i++)
    {
        if (a[i] == str)
            return i;
    }
}
```

√

```
    map<string,int> numbers = {
        {"zero",0},
        {"one",1},
        {"two",2},
        {"three",3},
        {"four",4},
        {"five",5},
        {"six",6},
        {"seven",7},
        {"eight",8},
        {"nine",9}
    };
```

√ √

补充

- Week 1 实验课 CodeReview 发现的问题
  - U 题，一定数额的人民币换成 100、50、10、5、2、1 元的钞票最少需要准备多少张纸币
  - 常量数组的使用

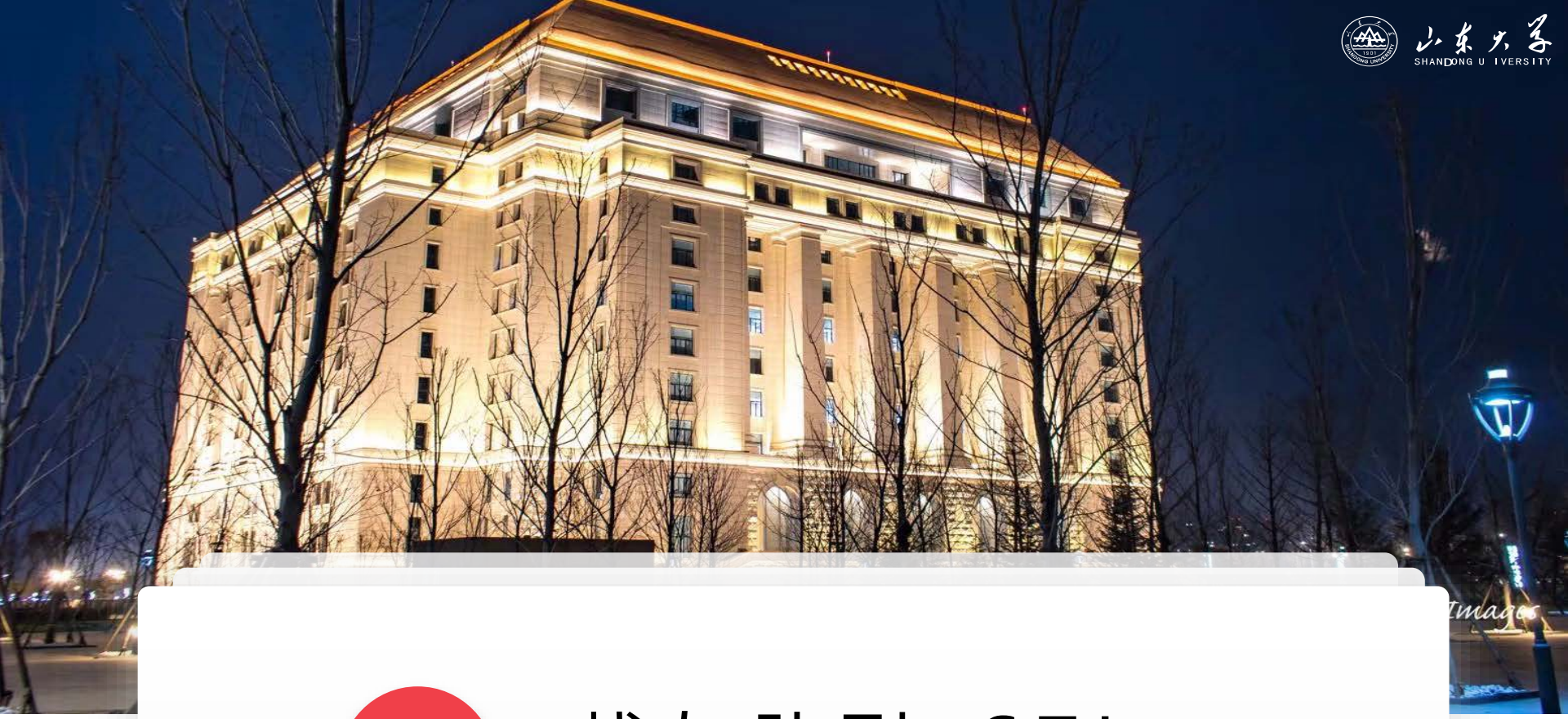
```
for(int i=0;i<n;i++)
{
    sum+=a[i]/100;
    a[i]=a[i]%100;
    sum+=a[i]/50;
    a[i]=a[i]%50;
    sum+=a[i]/10;
    a[i]=a[i]%10;
    sum+=a[i]/5;
    a[i]=a[i]%5;
    sum+=a[i]/2;
    a[i]=a[i]%2;
    sum+=a[i];
}
```

```
for(int i=0;i<n;i++)
{
    cin>>x;
    sum=x/100+x%100/50+x%100%50/10+x%100%50%10/5+x%100%50%10%5/2+x%100%50%10%5%2;
    count=count+sum;
}
```

const

```
int a[]={100,50,10,5,2,1};
int main(){
    for(int j=0;j<6;j++){
        c+=x/a[j];
        x%=a[j];
    }
    return 0;
}
```

√



1 栈与队列 STL

The STL of stack/queue

知识梳理

- 关于C++的STL，我们学习了哪些
- 想了解更多知识，可以查阅：http://www.cplusplus.com/

IO	scanf/printf cin/cout
<algorithm>库	sort/next_permutation/二分
vector	向量（不定长数组）
list	链表
string	字符串
stack	栈
queue	队列
priority_queue	优先队列（堆）
map	映射
set	集合
unordered_map	（了解 C++11）
unordered_set	（了解 C++11）

字符串

- string
  - 在C语言中，字符串就是字符数组，而不是像int/double那样的“一等公民”，使用起来处处受限
  - C++提供了#include <string>中的string类型，重载了很多运算符，使程序更加自然，简单。

```
string s;
cin >> s;   //字符串的声明 输入 输出
cout << s;

cout << s.size() << endl; //字符串长度

string a;

s += a;     //字符串连接字符串
s += '+';   //字符串连接字符

if (s == a)cout << "true" << endl; //判断两个串是否完全相等
if (s < a)cout << "true" << endl;  //判断两个串的字典序大小

for (int i = 0; i < s.size(); ++i)
    cout << s[i];   //遍历字符串的所有字符
```

栈与队列 STL

- stack
  - 栈，先进后出（后进先出）的数据结构
  - 概念数据结构上都有所涉及，这里主要关注 C++ 标准函数库的使用

“简单过一下”

```
/* 声明 */
stack<int> s;
stack<int, vector<int> > s; // 指定底层容器的栈
stack<int, list<int> > s;   // 指定底层容器的栈

/* 赋值 */
s.push(1); // 将1压栈

/* 访问 */
s.top();   // 访问栈顶

/* 清空 */
s.pop();                            // 弹出栈顶
while(!s.empty()) s.pop();          // 清空栈
for(int i=s.size(); i; i--) s.pop(); // 清空栈
```

栈与队列 STL

- queue / priority_queue
  - 队列，先进先出；优先队列，又称为“堆”

deque 可用 list 替代，也可自学

```
/* 声明 */
queue<int> q;

priority_queue<int> pq;                          // 优先队列，大根堆

// 大根堆 O(n) 线性构造
int a[] = {1,3,4,6,78,9}
priority_queue<int> pq(a, a+6);

priority_queue<int, vector<int>, greater<int> > pq; // 小根堆，结构体重载>方法
priority_queue<int, vector<int>, less<int> > pq;    // 大根堆，结构体重载<方法

/* 赋值 */
q.push(1); // 将1入队
pq.push(1);

/* 队列访问 */
q.front(); // 访问队首
q.back();  // 访问队尾

/* 优先队列访问 */
pq.top();  // 访问堆顶

/* 清空，二者是相同的 */
q.pop();                                  // 队首出队
pq.pop();                                 // 弹出堆顶
while(!q.empty()) q.pop();                // 清空队列
for(int i=q.size();i;i--) q.pop(); // 清空队列
```

栈与队列 STL

- queue / priority_queue
  - 优先队列怎么知道元素的大小判断方法？刚刚的例子因为使用了基本的数据类型，它们自带天然的大小判断方法。如果我们自己实现的复杂数据结构，则需要重写比较方法！
  - 结构体-优先队列 写法？ —— 复习结构体比较方法重写

```
struct P {
    int x, y, z;
    bool operator<(const P &p)const {
        if (x != p.x) return x < p.x; // 第一关键字升序
        if (y != p.y) return y > p.y; // 第二关键字降序
        return z < p.z;               // 第三关键字升序
    }
}Ps[1005];

priority_queue<P> heap;               // 大根堆

heap.push({1, 2, 3});
heap.push({2, 2, 3});
heap.push({1, 3, 4});
printf("%d\n", heap.top().x);         // 输出什么？  1 还是 2?
```

补充

- 结构体重载为什么只需要重载一个<号?

- 因为一个<号可以生成所有比较函数

```
bool operator > (const node_time &b)const{return b<*this;}
bool operator <= (const node_time &b)const{return !(b<*this);}
bool operator >= (const node_time &b)const{return !(*this<b);}
bool operator == (const node_time &b)const{return !(b<*this||*this<b);}
bool operator != (const node_time &b)const{return b<*this||*this<b;}
```

2 map/set及其底层原理

The principle of (unordered_) map/set

基于红黑树/哈希表的 map/set

下面先关注用法，再了解原理

- map
  - 概念：<key, value> 键值对。给一个 key，可以得到唯一 value
  - 基本使用：

```
/* 声明 */
map<int, bool> mp;

/* 赋值 */
mp[13] = true;

/* 查key是否有对应value */
if(mp[13]) printf("visited"); // 这里可以直接如此，而如果 value 类型非 bool 时需换成以下这种
if(mp.find(13) != mp.end()) printf("visited");

/* 遍历 */
for(map<int,int>::iterator it=mp.begin();it!=mp.end();it++)
    printf("%d %d\n", it->first, it->second); // 输出是升序的
for(map<int,int>::reverse_iterator it=mp.rbegin();it!=mp.rend();it++)
    printf("%d %d\n", it->first, it->second); // 输出是降序的

/* 遍历 c++11 特性 */
for(auto &entry : mp) printf("%d %d\n", entry.first, entry.second); // for range
```

基于红黑树/哈希表的 map/set

下面先关注用法，再了解原理

- map
  - 概念：<key, value> 键值对。给一个 key，可以得到唯一 value
  - 基本使用：

```
/* 清空 */
mp.erase(13); // 以键为关键字删除某该键-值对，复杂度是 log
mp.clear();

/* 常量map声明，而不是声明一个空的map随后在main中赋值 */
const map<char, char> mp({
    {'R', 'P'},
    {'P', 'S'},
    {'S', 'R'}
});
```

```
map<string,int> M;
string s1,s2,s3,s4,s5,s6;
int main()
{
    M["one"] = 1;
    M["two"] = 2;
    M["three"] = 3;
    M["four"] = 4;
    M["five"] = 5;
    M["six"] = 6;
    M["seven"] = 7;
    M["eight"] = 8;
    M["nine"] = 9;
    M["zero"] = 0;
```

```
/* 结构体使用map需要重写比较方法 */
struct Point {
    int x, y;
    bool operator<(const Point &p) const {
        return  x!=p.x ? x<p.x : y<p.y;
    }
};
int main()
{
    map<Point, bool> mp;
    printf("%d\n", mp[{1,5}]);
    mp[{1,5}] = true;
    printf("%d\n", mp[{1,5}]);
    return 0;
}
```

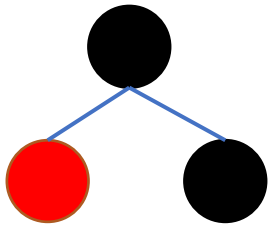
基于红黑树/哈希表的 map/set

- set
  - 概念：可以理解为数学意义上的“集合”， 三大特性？ （确定/互异/无序）
  - 对于一个元素，可以放入集合中；也可查找集合中是否有该元素（或删）

```
/* 声明 */
set<int> s;

/* 赋值 */
s.insert(9);   s.insert(4);   s.insert(6);   s.insert(4);  // set 中有3个元素

/* 遍历 */
if(s.find(1) != s.end()) printf("The element is in set");
for(set<int>::iterator it=s.begin();it!=s.end();it++) printf("%d", *it); // 全部遍历
for(auto &x : s) printf("%d\n", x);
int sz = s.size(); // 大小

/* 清空 */
s.erase(val); // 删除复杂度是logn
s.clear();

/* 二分 */
set<int>::iterator it = s.lower_bound(5); // 查找第一个大于等于 5 的元素，返回指针
if (it != set.end())    // 存在该元素
    printf("%d\n", *it);// 输出什么?
```

基于红黑树/哈希表的 map/set

- 红黑树 RB-Tree
  - C++ STL 中的 map/set 都是基于 RB-Tree 实现的，红黑树与之前学过的 AVL 树都是平衡树，但是红黑树不追求完全平衡，插入和删除的旋转次数较 AVL 树少，插入和删除的复杂度极优于 AVL 树。
  - 增删改查 的复杂度都是 log 级别
  - 并且，底层要求模板类<T>实现了比较方法



- 思考：
  - 利用红黑树来实现 map/set 好处是可以维护元素间的关系（有序性）
  - 但倘若不关心元素间的关系，map 的功能和数据结构上学的哈希表功能又一样，为什么不能要 O(1) 的增删改查性能?
  - —— 这便引入了基于哈希表的 unordered_map/set

基于红黑树/哈希表的 map/set

- unordered_map / unordered_set
  - 如果模板类<T> 是常用数据类型（int, char, bool 等），那么和 map / set 用法一样，只需要在声明时把 “map” 改为 “unordered_map”，就能把 O(logn) 的性能调为 O(1) ！
  - 提示：C++ 11 及其之后的版本才支持
  - 掌握 重写 == 方法、hash() 方法
    - 正如基于红黑树的 map/set 要求元素类型<T>重写比较方法
    - 基于哈希表的 unordered_map / unordered_set 在底层进行复杂元素的判断时，要求实现 判等方法和 hash() 方法

基于红黑树/哈希表的 map/set

- unordered_map / unordered_set
  - 掌握 重写 == 方法、hash() 方法

```
struct Point {
    int x, y;
    Point() {}
    Point(int _x, int _y):x(_x), y(_y) {}
    bool operator == (const Point &t) const {
        return  x==t.x && y==t.y;
    }
};
struct PointHash {
    std::size_t operator () (const Point &p) const {
        return  p.x * 100 + p.y;   // 如果数据范围 x,y<100, hash 方法可以这样写
    }
};
int main()
{
    unordered_map<Point, bool, PointHash> mp;
    printf("%d\n", mp[{1,5}]);
    mp[{1,5}] = true;
    printf("%d\n", mp[{1,5}]);
    unordered_set<Point, PointHash> st;
    return 0;
}
```

基于红黑树/哈希表的 map/set

- unordered_map / unordered_set
  - 了解 哈希因子（负载因子）调参
    - 需要思考，为什么 unordered_map 比 map 时间复杂度优，无非以下几点：
      - 取消维护元素间大小关系
      - 更大的常数耗时（hash方法）
      - 空间换时间
    - 负载因子 = 元素数量 ÷ 桶数量，一般 0.75 即可

```
unordered_map<Point, bool, PointHash> mp;
/* 设置最大的负载因子 */
mp.max_load_factor(0.75);
printf("桶数量  %d     负载因子 %f\n", mp.bucket_count(), mp.load_factor());
mp[{1,5}] = true;   mp[{2,4}] = true; mp[{3,3}] =  true;
printf("桶数量  %d     负载因子 %f\n", mp.bucket_count(), mp.load_factor());
```

基于红黑树/哈希表的 map/set

- 总结

	map/set	unordered_map/set
底层	红黑树	哈希开链法
元素间关系	元素有序，可从小到大遍历	元素无序，可自然遍历
需要重写方法	比较方法	判等方法、hash()方法
单次操作复杂度	log 级别	最好 O(1) 最坏 O(n)
总体复杂度	复杂度较稳定	大部分情况复杂度优 但“常数比较大”（hash()方法的常数耗时、哈希表的建立）

- map 的用法主要有三个 （了解）
  - 离散化数据
  - 判重 (set也行)
  - 需要 logn 级别的 insert/delete 性能，同时维护元素有序（后续课程涉及）

3 广 度 优 先 搜 索

Breadth First Search

广度优先搜索

- BFS
  - 在 DS 课上已经学过了 BFS 的概念，所以主要关注：
    - ACM 上利用了 STL<queue/priority_queue> 一般的解题代码框架
    - CSP 中 BFS 题的考察
    - 隐式图问题
    - BFS 的优化

搜索就是扩散

广度优先搜索

- BFS 引例 —— 迷宫问题：
  - 一个n*m的迷宫；

    有的格子里有障碍物，不能走；

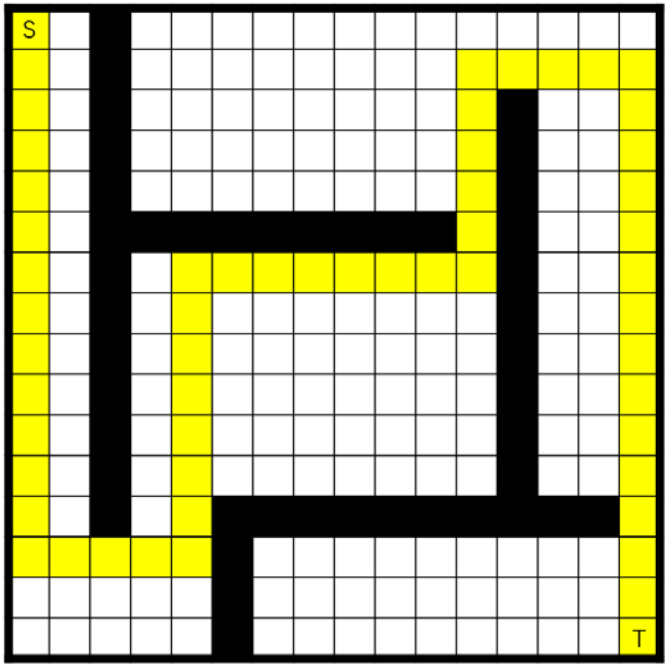
    有的格子是空地，可以走；
  - 给定一个迷宫，求从左上角走到右下角最少需要走多少步(数据保证一定能走到)。只能在水平方向或垂直方向走，不能斜着走。

样例输入

01000
01010
01010
00010

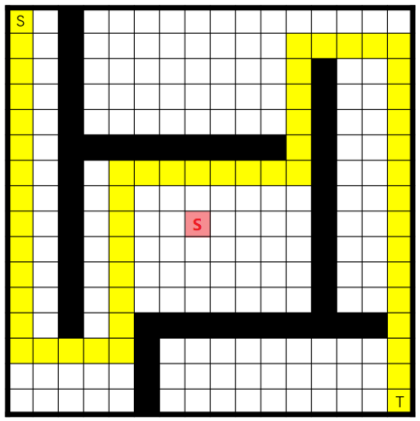
样例输出

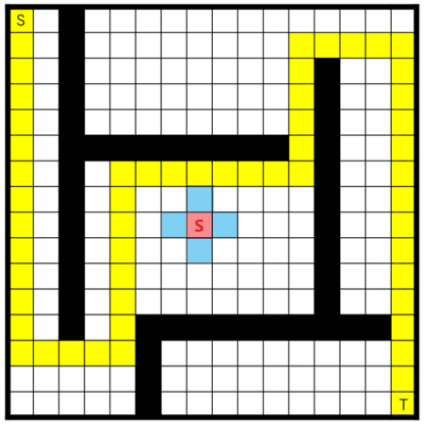
13

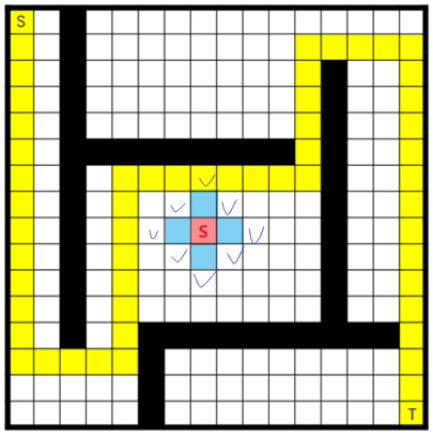


广度优先搜索

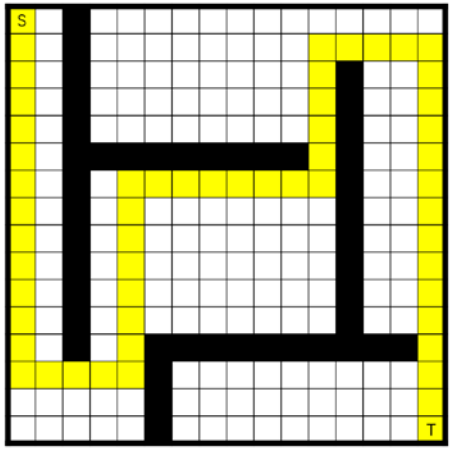
- BFS 引例 —— 迷宫问题：
  - 类似树的按层遍历。
  - 1. 访问初始点(sx,sy)，并将其标记为已访问过。
  - 2. 访问(sx,sy)的所有未被访问过可到达的邻接点，并均标记为已访问过，将这些点加入到队列中。
  - 3. 再按照队列中的次序，访问每一个顶点的所有未被访问过的邻接点，并均标记为已访问过，加入到队列中，依此类推。
  - 4. 直到到达终点或者图中所有和初始点Vi有路径相通的顶点都被访问过为止。







广度优先搜索



- BFS 引例 —— 迷宫问题：
  - 代码

```
queue<point> Q;
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};

int bfs() {
    Q.push(point(sx, sy));
    vis[sx][sy] = true;
    dis[sx][sy] = 0;
    while (!Q.empty()) {
        point now = Q.front();
        Q.pop();
        if (vis[now.x][now.y])break;
        for (int i = 0; i < 4; ++i) {
            int x = now.x + dx[i], y = now.y + dy[i];
            if (x >= 1 && x <= n && y >= 1 && y <= m
                && !vis[x][y] && !a[x][y]) {
                dis[x][y] = dis[now.x][now.y] + 1;
                vis[x][y] = 1;
                Q.push(point(x, y));
            }
        }
    }
    return dis[tx][ty];
}
```

广度优先搜索

- BFS 解决一般问题的框架 总结：
  - vis 数组，用来记录某个状态下的最优结果，以免反复到达，一般是一维或二维，有可能使用高维数组（当状态比较复杂，比如增加了时间维度）
  - queue 队列，用于记录“层层”拓展的中间节点
  - dis 可有可无，上例中使用了这个来记录到某个点的最短距离，但是可以和 vis 数组进行合并
  - While(!Q.empty() 循环
    - 取点
    - 拓展周边节点（小技巧？）
    - 特判合法性，加入队列

广度优先搜索

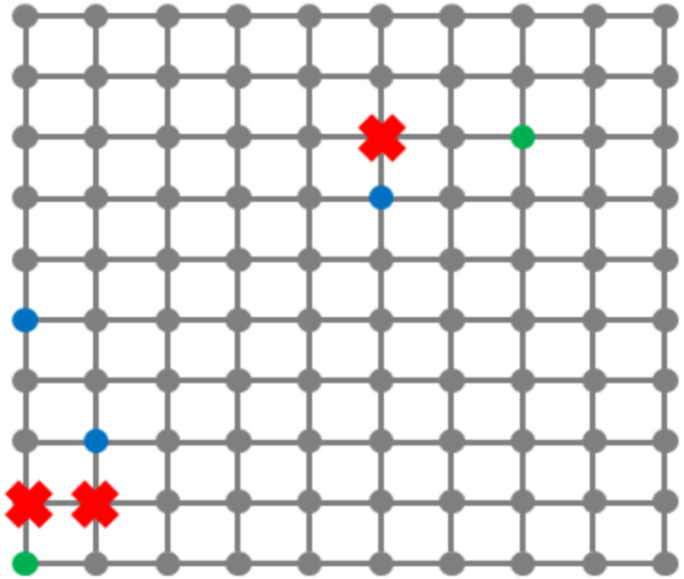
- CSP 中 BFS 的考察
  - CSP201409-T4 最优配餐

问题描述

栋栋最近开了一家餐饮连锁店，提供外卖服务。随着连锁店越来越多，怎么合理的给客户送餐成为了一个急需解决的问题。

栋栋的连锁店所在的区域可以看成是一个n×n的方格图（如下图所示），方格的格点上的位置上可能包含栋栋的分店（绿色标注）或者客户（蓝色标注），有一些格点是不能经过的（红色标注）。

方格图中的线表示可以行走的道路，相邻两个格点的距离为1。栋栋要送餐必须走可以行走的道路，而且不能经过红色标注的点。



送餐的主要成本体现在路上所花的时间，每一份餐每走一个单位的距离需要花费1块钱。每个客户的需求都可以由栋栋的任意分店配送，每个分店没有配送总量的限制。

现在你得到了栋栋的客户的需求，请问在最优的送餐方式下，送这些餐需要花费多大的成本。

广度优先搜索

- CSP 中 BFS 的考察
  - CSP201409-T4 思路
    - 若只有一个起点的话，那么成本 = 每个点的距离 * 每个点的客户需求量
    - 方案1：分别对每个起点点进行一次BFS，距离 = 所有点为起点的距离的最小值
    - 方案2：直接进行一次BFS，把起始的所有点都一次性添加进队列中，dis初值赋为0（其余点的dis值为inf），最终得到的dis值就是到该点的最小距离。
      - 更好的理解方式：加一个超级源点 -> 到各个点的代价是 0

广度优先搜索

- CSP 中 BFS 的考察
  - CSP201604-T4 游戏

问题描述

小明在玩一个电脑游戏，游戏在一个$n \times m$的方格图上进行，小明控制的角色开始的时候站在第一行第一列，目标是前往第n行第m列。

方格图上有一些方格是始终安全的，有一些在一段时间是危险的，如果小明控制的角色到达一个方格的时候方格是危险的，则小明输掉了游戏，如果小明的角色到达了第n行第m列，则小明过关。第一行第一列和第n行第m列永远都是安全的。

每个单位时间，小明的角色必须向上下左右四个方向相邻的方格中的一个移动一格。

经过很多次尝试，小明掌握了方格图的安全和危险的规律：每一个方格出现危险的时间一定是连续的。并且，小明还掌握了每个方格在哪段时间是危险的。

现在，小明想知道，自己最快经过几个时间单位可以达到第n行第m列过关。

输入格式

输入的第一行包含三个整数n，m，t，用一个空格分隔，表示方格图的行数n、列数m，以及方格图中有危险的方格数量。

接下来t行，每行4个整数r，c，a，b，表示第r行第c列的方格在第a个时刻到第b个时刻之间是危险的，包括a和b。游戏开始时的时刻为0。输入数据保证r和c不同时为1，而且当r为n时c不为m。一个方格只有一段时间是危险的（或者说不会出现两行拥有相同的r和c）。

输出格式

输出一个整数，表示小明最快经过几个时间单位可以过关。输入数据保证小明一定可以过关。

样例输入

```
3 3 3
2 1 1 1
1 3 2 10
2 2 2 10
```

样例输出

```
6
```

广度优先搜索

- CSP 中 BFS 的考察
  - CSP201604-T4 思路
    - 该题是 BFS 的另一种变形，这里障碍物并不是每时每刻都存在的，而是只在一个连续的区间存在。
    - 首先记录的时候就应该多记录一维时间，然后才能判断是否能够扩展到其它位置
    - 注意这里不要直接使用 vis[][] 数组判断这一个点是否经过，因为一个可以会重复经过，要使用 vis[][][] 多记录一维时间
    - 本题目样例是较好的，它可以提示我们不能只是记录这个点是否访问过。

广度优先搜索

- 隐式图问题
  - 隐式图是仅给出初始结点、目标结点以及生成子结点的约束条件（题意隐含给出），要求按扩展规则应用于扩展结点的过程，找出其他结点，使得隐式图的足够大的一部分编程显式，直到包含目标结点为止。
  - 倒水问题
    - 给你两个容器，容量分别为A， B ，问是否能够经过有限的步骤倒水，得到容量为 C 的水

广度优先搜索

- 隐式图问题
  - 倒水问题
    - 给你两个容器（容量分别为A， B）、一台饮水机，问是否能够经过有限的步骤倒水，得到容量为 C 的水
    - 对于每一种状态，都有以下几种转移方式：
      - A倒入B
      - B倒入A
      - A倒空/满
      - B倒空/满
    - 可以转移到不同的状态。
    - 对于每一种状态，我们需要记录是否已经被访问过了（BFS的过程）
    - 可以直接使用高维数组记录（本题），有时需要用 map/set 或哈希算法来记录状态

代码： https://paste.ubuntu.com/p/VpgG59MPXS/

广度优先搜索

- BFS 优化 —— 双向广度优先搜索 （了解）
  - 在广度优先搜索的基础上进行优化，采用双向搜索的方式，即从起始节点向目标节点方向搜索，同时从目标节点向起始节点方向搜索。
  - 特点：
    - 1.双向搜索只能用于广度优先搜索中。
    - 2.双向搜索扩展的节点数量要比单向少的多。

广度优先搜索

- BFS 优化 —— 双向广度优先搜索 （了解）
  - 用同一个队列来保存正向和逆向扩展的结点。开始时，将起点坐标和终点坐标同时入队列。这样，第1个出队的坐标是起点，正向搜索扩展队列；第2个出队的坐标是终点，逆向搜索扩展队列。两个方向的扩展依次交替进行。
  - 简单修改vis[][]数组元素的置值方法即可。初始时，vis数组的全部元素值为0，由正向扩展来的结点的vis对应元素值置为1，由逆向扩展来的结点的vis对应元素值置为2。
  - 设当前结点为cur，由cur可以扩展出新结点next。若vis[][]==0，则next结点未访问过，将next结点入队并进行相应设置；若vis[][]!=0且cur和next的vis值不相同，表明正向反向相遇，搜索成功。

总结


Week2
栈与队列 STL
stack
声明、插入、访问、删除
queue
声明、插入、访问、删除
priority_queue
声明、插入、访问、删除
O(n) 线性构造
结构体重写比较方法
基于红黑树/哈希表的 map/set
map/set
声明、插入、访问、删除
底层是红黑树，需要重写比较方法
O(logn)
unordered_map/unordered_set
声明、插入、访问、删除
底层是哈希表，需要重写判等方法和hash方法
好O(1) 坏O(n)
了解 负载因子
广度优先搜索
一般解题框架
CSP 中对 BFS 的考察
CSP201409-T4
CSP201609-T4
隐式图问题
例子：倒水问题
了解：双向BFS优化

感谢收听

Thank You For Your Listening