基本数据结构与简单搜索技术

$67Plus$

南京航空航天大学

2024/10/26

栈和stack

栈是基本的数据结构之一，特点是“先进后出”. 例如一盒泡腾片，最先放进盒子的药片位于最底层，最后被拿出来

stack

stack是*STL*中的一种container

```
#include <stack>          //头文件
stack<Type> s;            //定义栈,Type为数据类型,例如int,float等
s.push(item);             //将item推入栈顶
it = s.top();             //获取栈顶元素并赋值给it
s.pop();                  //弹出栈顶元素,但没有返回值
s.size();                 //返回栈中元素的个数
s.empty();                //检查栈是否为空,为空则返回true,否则返回false
```

队列和queue

队列也是基本数据结构之一，特点是“先进先出”. 例如现实生活中排队，先来的人先得到服务.

queue

和stack一样，queue也是*STL*中的一种container

```
#include <queue>          //头文件
queue<Type> q;            //定义栈,Type为数据类型,例如int,float等
q.push(item);             //将item放进队列
it = q.front();           //获取队头元素并赋值给it
q.pop();                  //弹出队头元素,但没有返回值
q.size();                 //返回队列中元素的个数
q.empty();                //检查队列是否为空,为空则返回true,否则返回false
```

其他常用基本数据结构

- **优先队列priority_queue**: 带有排序功能的queue
- **集合set**: 每个元素只出现一次,且是排好序的
- **map<key,value>**: 实现从key到value的一个映射
- **链表list**(很少用)

> **除了list外, 各种数据结构(尤其是map和set)在竞赛中的应用非常广泛**

简单搜索技术

搜索技术是“暴力法”思想的具体实现

> **暴力法(Brute force): 把所有可能得情况都罗列出来, 然后逐一检查, 从中找到答案**

深度优先搜索(DFS)和广度优先搜索(BFS)是最基本的两种搜索方法

算法思路

想象老鼠走迷宫问题, 老鼠在每个岔路口有左右手两条路可以走

- 一只老鼠走迷宫. 这只老鼠在每个路口都选择先走右边(或左边),能走多远就走多远, 直到碰壁没有办法继续走, 就**回退一步**,这次走左边(或右边), 继续往下走. 这个方法可以走遍**所有**的路,而且**不会重复**. 这个思路就是**DFS**
- 一群老鼠走迷宫. 假设有无限只老鼠进入迷宫,在每个路口分出一部分老鼠探索所有**没走过的路**, 走某条路的老鼠如果碰壁没法继续走, 就**原地停下**. 如果走到的路口已经有其他老鼠探索过了, 也**原地停下**.很显然,所有的道路也都会走到, 且不会重复. 这个思路就是**BFS**

迷宫寻路

机器猫被困在一个矩形迷宫里。迷宫可以视为一个 $n \times m$ 矩阵，每个位置要么是空地，要么是墙。机器猫只能从一个空地走到其上、下、左、右的空地。
机器猫初始时位于 $(1, 1)$ 的位置，问能否走到 (n, m) 。

输入

```
3 5
.##.#
.#...
...#.
```

输出

```
Yes
```

```
tips:
bool check(int x, int y) {
    return x >= 1 && y >= 1 && x <= n && y <= m && visited[x][y] == false && graph[x][y] == '.';
}// 用于判断坐标合法性
int dx[4] = { 1,0,-1,0 };
int dy[4] = { 0,1,0,-1 };//分别对应下右上左四个移动方向
```

DFS与递归

用DFS的思路, 我们考虑这道题

我们用$visited[i][j]$记录(i,j)的位置有没有被走到过, 以下是DFS的算法执行过程

1. 令$visited[1][1] = true$, 表示$(1,1)$的位置已经走到过
2. 右下左上四个方向,按顺时针顺序(或者你喜欢的顺序)选一个能走的方向走一步(不能走的情况有障碍物或**坐标越界**等)
3. 假设新的位置为(i,j), 令$visited[i][j] = true$
4. 重复2的过程继续前进.
5. 如果在(x,y)处无路可走, 则回退一步,不需要令$visited[x][y] = false$, 因为已经走过的点无须重复遍历
6. 重复以上过程, 如果过程中出现$visited[n][m] == true$, 则说明能走到, 输出Yes, 否则输出No

我们用递归函数实现上述过程，就是dfs的代码

```
void dfs(int x, int y) {
    visited[x][y] = true;
    if (x == n && y == m) { //达成返回条件就返回
        ok = true;
        return;
    }
    for (int i = 0; i < 4; i++) {
        int tox = x + dx[i];
        int toy = y + dy[i];
        if (check(tox, toy)) dfs(tox, toy);//递归
        if (ok) return; //达成条件直接返回不再继续遍历
    }
}
```

BFS与队列

同样用$visited[i][j]$记录(i,j)的位置有没有被走到过，以下是BFS的算法执行过程

1. $(1,1)$进队，当前队列是$(1,1)$
2. 队首元素$(1,1)$出队，他的两个邻居$(1,2)$和$(2,1)$进队,可以理解为从$(1,1)$扩散到周围坐标
3. 对于队列中每个元素逐一处理，将其周围**没有进队**的元素进队(记得check坐标合法性)
4. 重复2,3操作，直至队列为空时算法结束，此时若$visited[n][m] == true$，则可以到达，反之不行

用STL中的queue实现上述过程

```
bool bfs(int x, int y) {
    queue<pair<int, int>> q;//pair换成struct存xy值也可以
    q.push({ x,y });
    visited[1][1] = true;
    while (!q.empty()) {
        int nowx = q.front().first, nowy = q.front().second; // 获取队首元素
        q.pop(); //弹出队首元素
        for (int i = 0; i < 4; i++) {
            int tox = nowx + dx[i];
            int toy = nowy + dy[i];
            if (check(tox, toy)) {
                q.push({ tox,toy });//满足条件的新元素入队
                visited[tox][toy] = true;// 入队即视为访问过,即使还在队列中没有处理
            }
        }
    }
    if (visited[n][m]) return true;
    else return false;
}
```

举一反三

题面同迷宫寻路

- 给出任意两个位置询问是否能从a走到b
- 机器猫只能向右或向下移动, 求:
  - 任意两点之间的路径数
  - 任意两点之间的最短路径长度