2019春

程序设计实习(II): 算法设计

第十六讲 深度优先搜索

刘家瑛

liujiaying@pku.edu.cn

课前多叽歪

- 期中考试
  - 选择题: 1题1分, 共计20分
  - 填空题: 给分方式 (×0.8)

11	**100**	**5**	**70**
10	**95**	**4**	**65**
9	**90**	**3**	**60**
8	**85**	**2**	**50**
7	**80**	**1**	**40**
6	**75**		

- 翻篇继续干

课前多叽歪

- 抓紧一切在机房上机练习的机会
  - 还剩3次上机机会: 5月26日, 6月2日 和 6月9日

程序设计实习 / 2019 程序设计实习之周末上机练习一 (刘老师班) 已经结束

题目 | 排名 | 状态 | 统计 | 提问

比赛已经结束

2019-05-19 15:00:00
开始时间

2019-05-19 17:00:00
结束时间

题目ID	标题	通过率	通过人数	尝试人数
A	浮点数求高精度幂	0%	0	1
B	Integer Inquiry	100%	9	9
C	Communication System	60%	3	5
D	判断闰年	100%	14	14
E	生理周期	92%	11	12
F	完美立方	100%	14	14
G	画家问题	100%	7	7
H	恼人的青蛙	50%	1	2
I	放苹果	100%	11	11
J	古代密码	83%	5	6

- 准备习题讲解, 鼓励参与, 讲述自己的心路历程
  - 邮件宋思捷, 报名主讲题目, 先到先得 [今日12点后]

主要内容

- 生活中的搜索问题 —— 广搜 & 深搜
- 搜索 Vs. 枚举 / 递归
- 深度优先搜索的基本概念
- 例题1: 城堡问题 [入门题目]
- 例题2: 寻路问题 [入门问题]
- 例题3: 拯救少林神棍 [郭老师史上最得意作品]

生活中的搜索问题

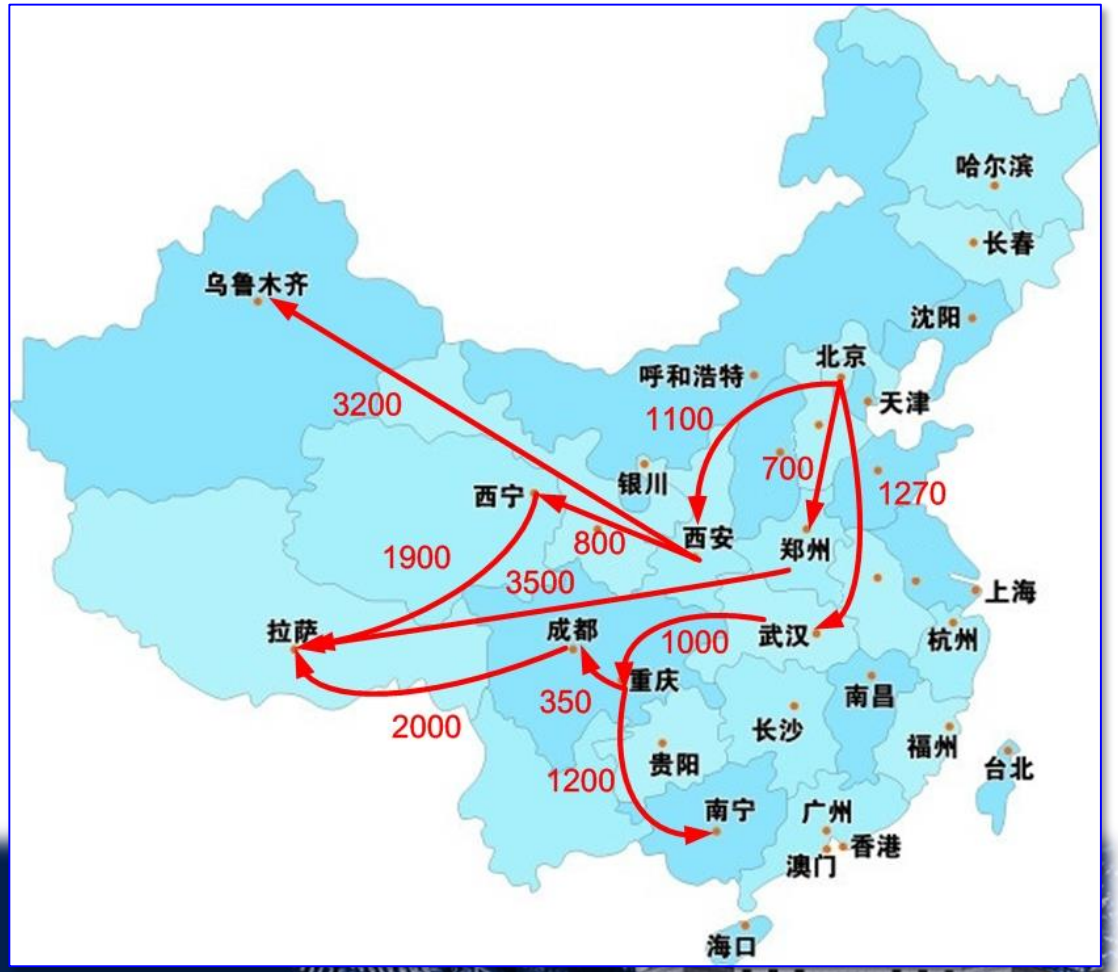
■ 例题: 行程定制问题

生活中的搜索问题

行程定制问题

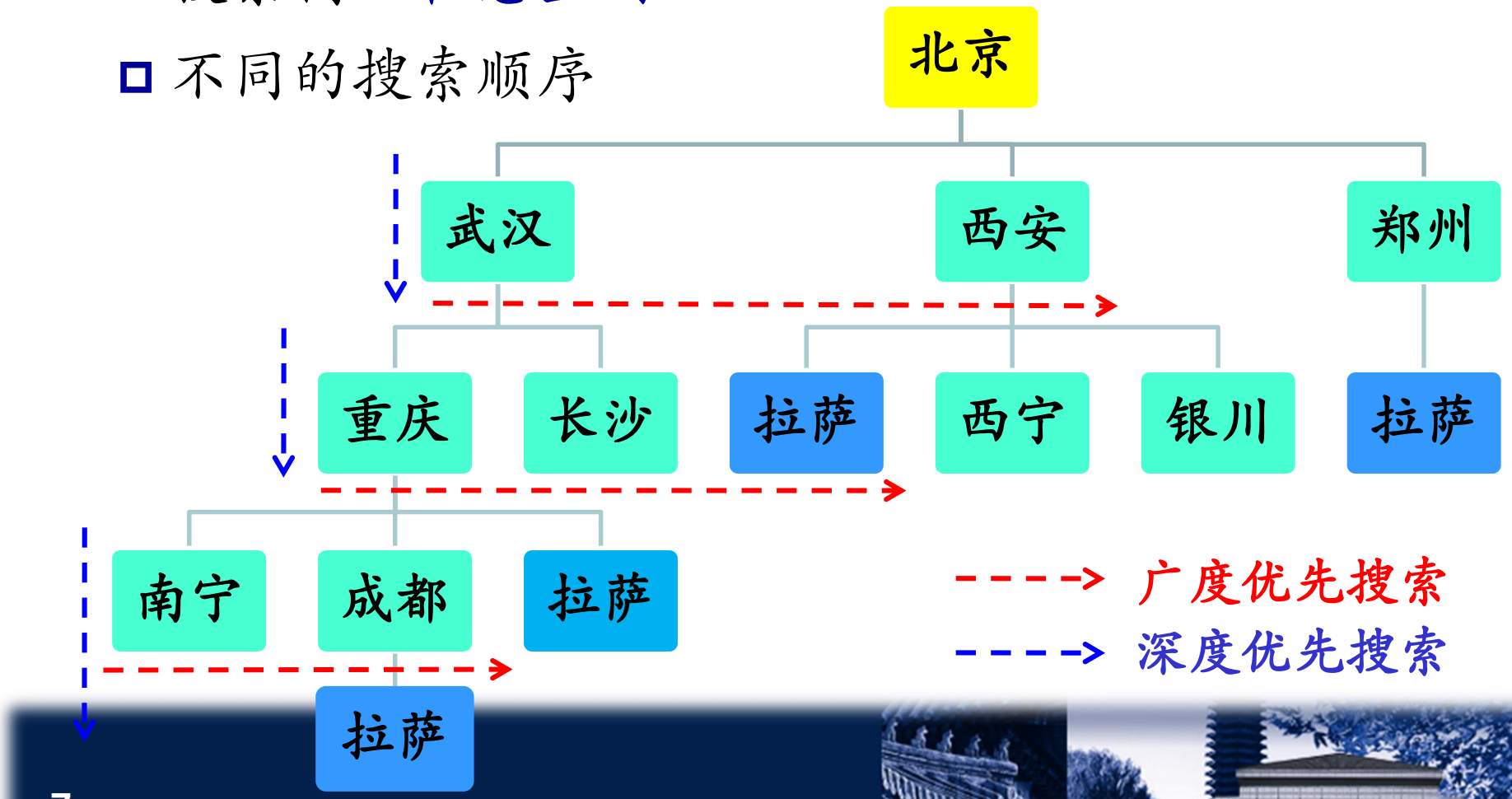
行程表与图的表示

航班	距离
北京--西安	1100公里
西安--乌鲁木齐	3200公里
北京--郑州	700公里
郑州--拉萨	3500公里
北京--武汉	1270公里
武汉--重庆	1000公里
重庆--成都	350公里
成都--拉萨	2000公里
西安--西宁	800公里
重庆--南宁	1200公里
西宁--拉萨	1900公里



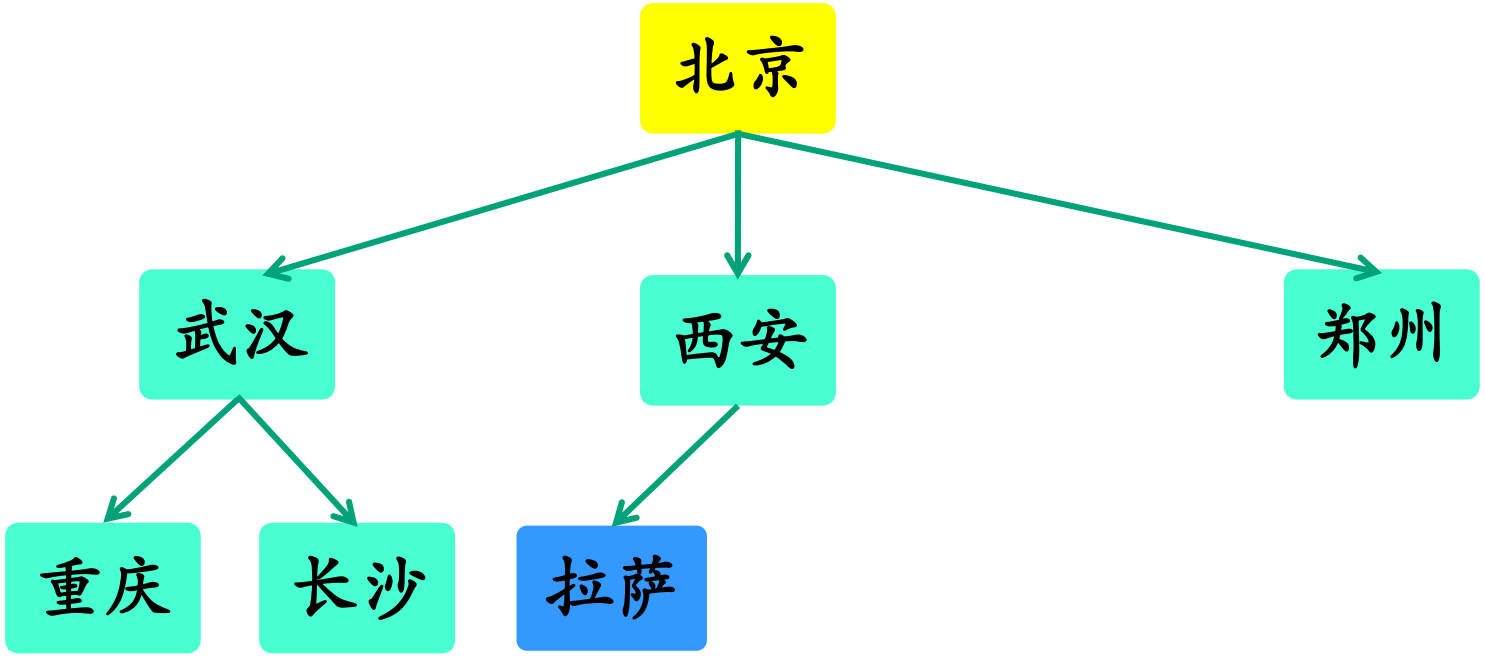
生活中的搜索问题

- 行程定制问题
  - 搜索树→状态空间
  - 不同的搜索顺序



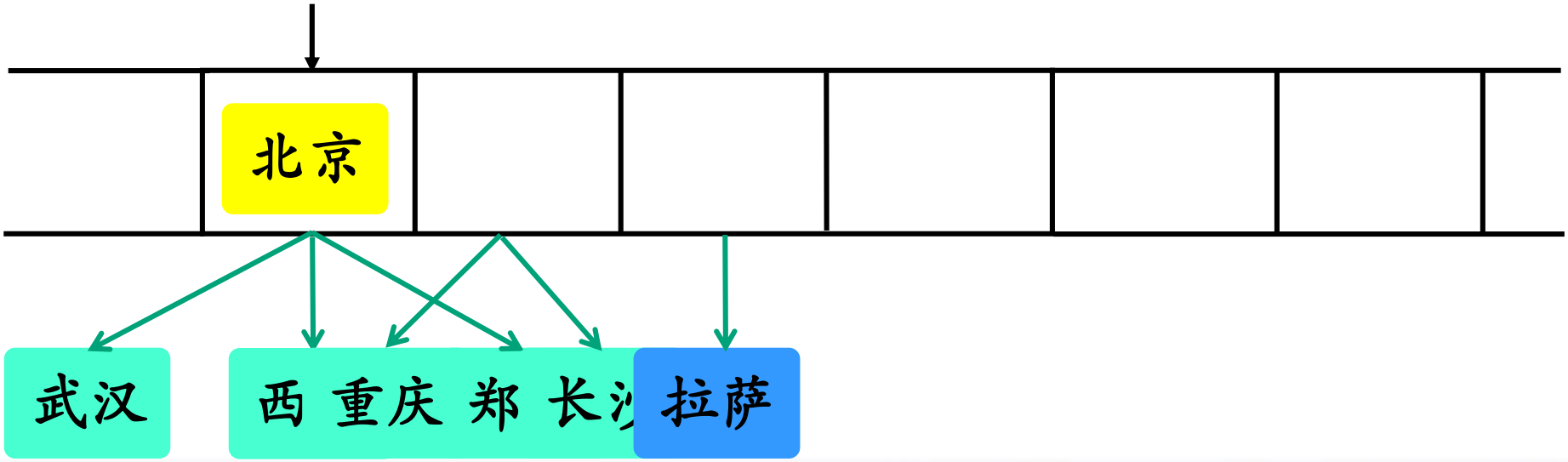
广度优先搜索

- 广度优先搜索 (Breadth-First-Search, BFS)
  - 优先扩展浅层结点, 逐渐深入



广度优先搜索

- 广度优先搜索
  - 用**队列**保存待扩展的结点
  - 从队首队取出结点，扩展出的新结点放入队尾，直到找到目标结点 [问题的解]



广度优先搜索

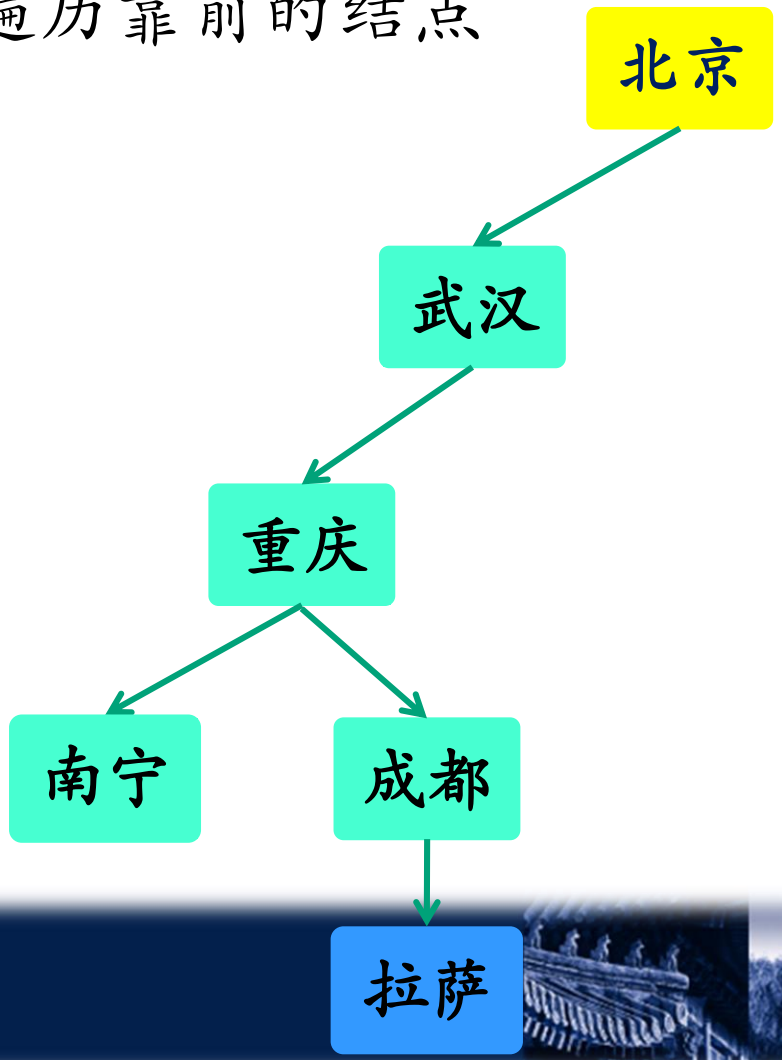
- 广度优先搜索—代码框架

```
BFS(){
        初始化队列;
        while( 队列不为空 & 未找到目标结点 ){
                取队首结点扩展,并将扩展出的结点放入队尾;
                必要时要记住每个结点的父结点;
        }
}
```

Pre	Data

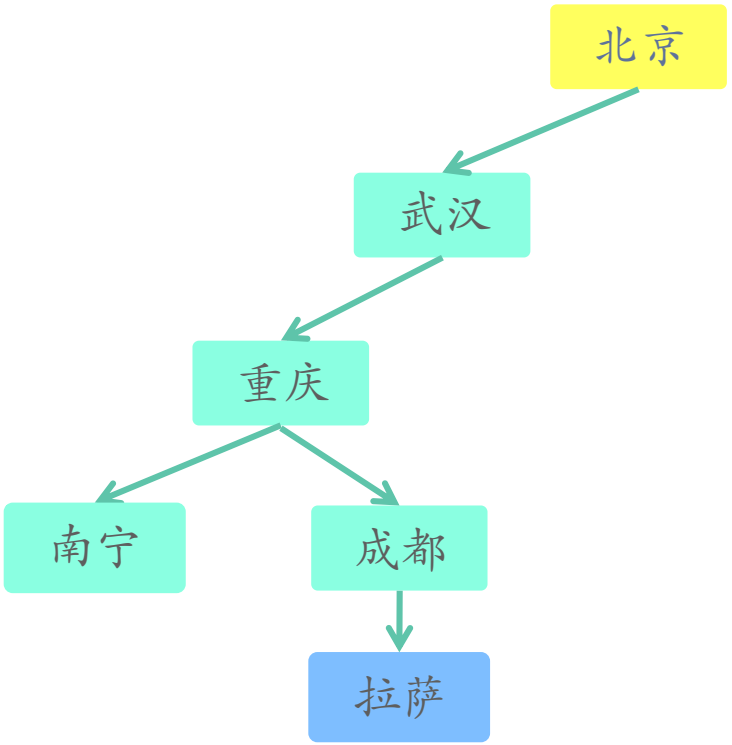
深度优先搜索

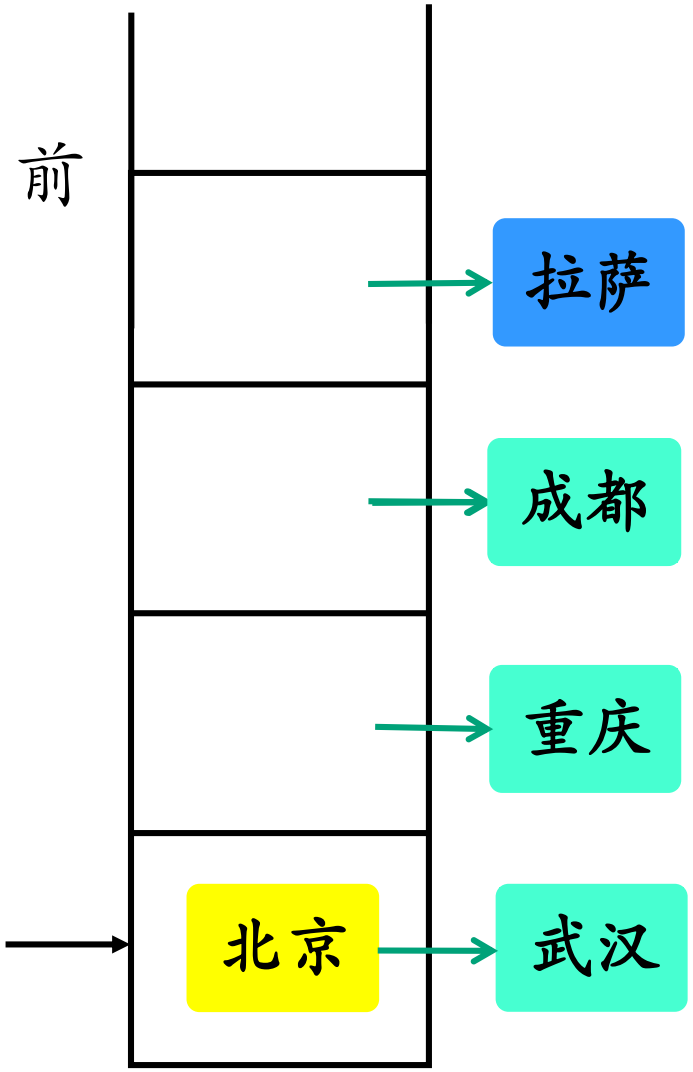
- **深度优先搜索** (Depth-First-Search, DFS)
  - 优先深入遍历靠前的结点



深度优先搜索

- 深度优先搜索
  - -可以用堆栈实现
  - -在栈中保存从起始结点到当前结点的路径上的所有结点





深度优先搜索

- 深度优先搜索—代码框架

```
DFS(){
        初始化栈;
        while( 栈不为空 & 未找到目标结点 ){
                取栈顶结点扩展,扩展出的结点放回栈顶;
        }
        ……
}
```


枚 举

- 枚 举
  - 划定解的存在空间
  - 对该空间的元素逐个判断
- **例1:** 求出A-I这九个字母对应的数字 (1-9), 使得下式成立 (一一对应)

$$\begin{array}{r} \mathbf{ABCD} \\ \times \quad \mathbf{E} \\ \hline \mathbf{FGHI} \end{array}$$

枚　举

ABCD

×　　E

FGHI

- **解法:**
  - 枚举ABCDE的值, 计算乘积, 判断是否符合要求

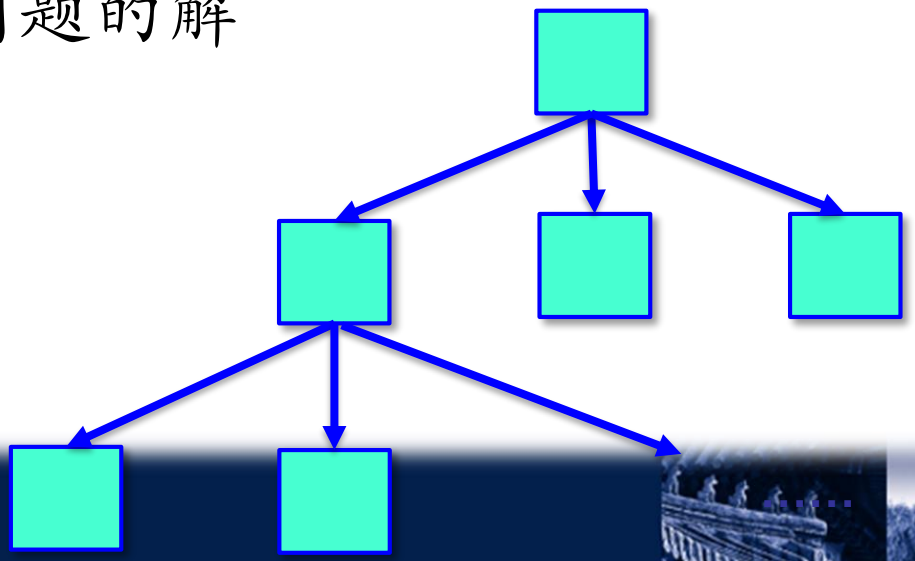
枚举与搜索

- **枚举 Vs. 搜索**
  - 枚举

    逐一判断所有可能的方案是否为问题的解;
  - 搜索: 高级枚举

    有顺序有策略地枚举状态空间中的结点,

    寻找问题的解



搜索：复杂的，高级的枚举

- **枚举：**解空间中的每个元素是一个**动作的集合** F
  - 将初态S_0 → 另一个状态$F(S_0)$
  - F中**各动作的执行顺序**不影响$F(S_0)$
  - 如果$F(S_0)$是符合要求的S*，那么F是真解，否则是伪解
- **搜索：**解空间的每个元素是一个**动作的序列** F
  - 将初态S_0 → 另一个状态$F(S_0)$
  - 如果$F(S_0)$是符合要求的S*，那么F是真解，否则是伪解
  - 有一个规则，确定在每个状态S下，分别有**哪些动作可供选择**
  - 采用**递归**的办法，产生每个动作序列

搜索与递归

- **搜索:** 有顺序有策略地产生解空间的元素
  - 每个解空间的元素表现为一定**动作的执行轨迹** (trace of actions)
- 采用**递归**的策略产生解空间的元素, 出口条件
  - 轨迹已经达到终点: 真解, 或者伪解
  - 轨迹不可能导出真解

搜索的过程

- **两个状态的集合**
  - α: 未处理完的状态
  - β: 已处理的状态
- **状态的处理:** 有顺序的尝试备选动作, 每一次的尝试都演化出另一个状态
  - 已处理的状态: 全部备选动作都已经尝试
- **树结构:** 状态之间的演化关系
- **递归的出口**
  - α为空
  - 演化出目标状态S^*
  - 演化出的状态属于$\alpha \cup \beta$

影响搜索效率的因素

- 两个状态的集合
  - α: 未处理完的状态
  - β: 已处理的状态
- **判重:** 每次演化出一个状态s时, s是否属于α或者β
- **剪枝:** 状态s的任意演化结果是否都属于β
- 演化出来的状态数量: $\alpha \cup \beta$的大小

深度优先搜索

- **两个状态的集合**
  - α: 未处理完的状态
  - β: 已处理的状态
- 从α中选择**被演化状态的原则**:离初态s_0最远的状态s
  - s_0到s的距离: 从s_0到达s使用的动作数量
- **实现方法**: 用stack表示α
  - 每次取stack顶部的状态演化
  - 每次演化出的状态s若不属于β, 则s将压入stack顶部

深度优先搜索

- 深度优先搜索
  - 可以用**堆栈**实现，在栈中保存从起始结点(状态)到当前结点的路径上的所有结点
  - 一般用递归实现

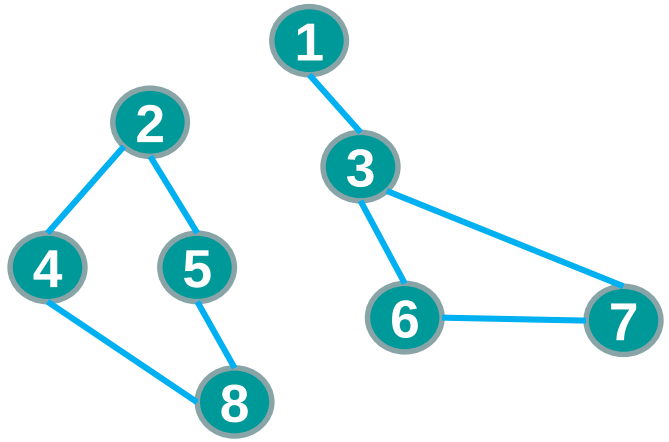
深度优先搜索

入门: 城堡问题

问题的各状态之间的转移关系描述为一个“图”

→深度优先搜索遍历整个图的框架为：

```
Dfs(v) {
    if( v访问过 )
        return;
    将v标记为访问过;
    对和v相邻的每个点u: Dfs(u);
}
int main() {
    while(在图中能找到未访问过的点 k)
        Dfs(k);
}
```



搜索顺序：

2-4-8-5

1-3-6-7

24

例题：城堡问题 (百练2815)

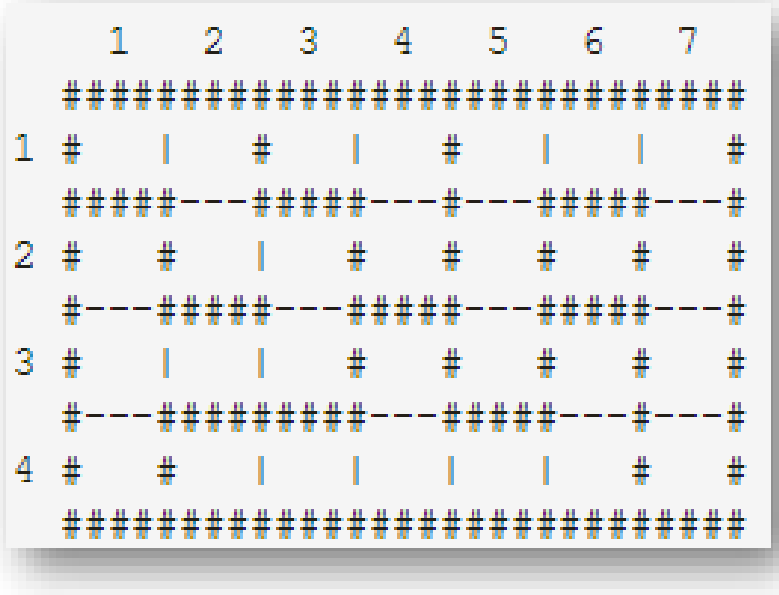
- 右图是一个城堡的地形图
- 请你编写一个程序，

  计算城堡一共有**多少房间**，

  最大的房间有多大
- 城堡被分割成m×n

  (m≤50, n≤50)个方块，

  每个方块可以有0~4面墙



= Wall
| = No Wall
- = No Wall

25

输入输出

- **输入**
  - 程序从标准输入设备读入数据
  - 第一行是两个整数，分别是南北向，东西向的方块数
  - 接下来的输入行，每个方块用一个数字(0≤p≤50)描述
    - 用一个数字表示方块周围的墙
    - 1表示西墙，2表示北墙，4表示东墙，8表示南墙
    - 每个方块用代表其周围墙的数字之和表示
    - 城堡的内墙被计算两次：

      方块(1,1)的南墙同时也是方块(2,1)的北墙
  - 输入的数据保证城堡至少有两个房间

26

输入输出

- **输出**
  - 城堡的房间数，城堡中最大房间所包括的方块数
  - 结果显示在标准输出设备上

27

- 样例输入

4

7

11 6 11 6 3 10 6

7 9 6 13 5 15 5

1 10 12 7 13 7 5

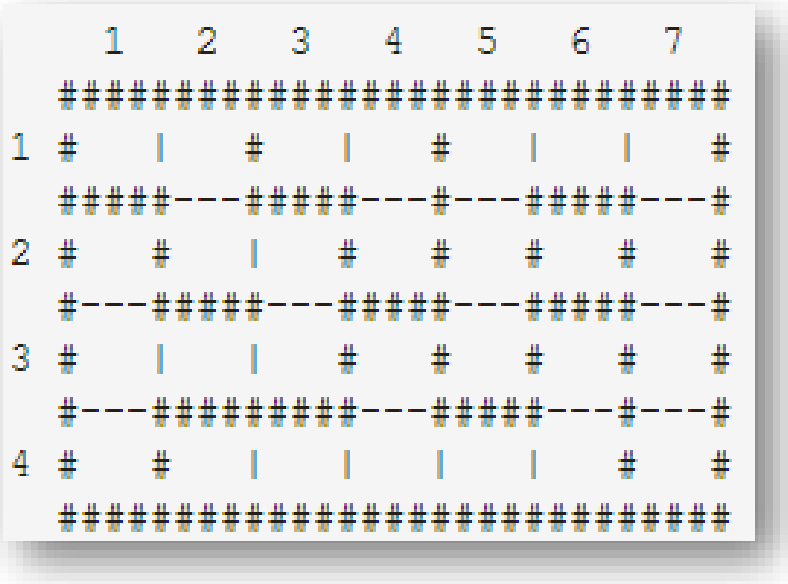
13 11 10 8 10 12 13

- 样例输出

5

9

- 1表示西墙, 2表示北墙, 4表示东墙, 8表示南墙
- 每个方块用代表其周围墙的数字之和表示



= Wall
| = No Wall
- = No Wall

28

解题思路

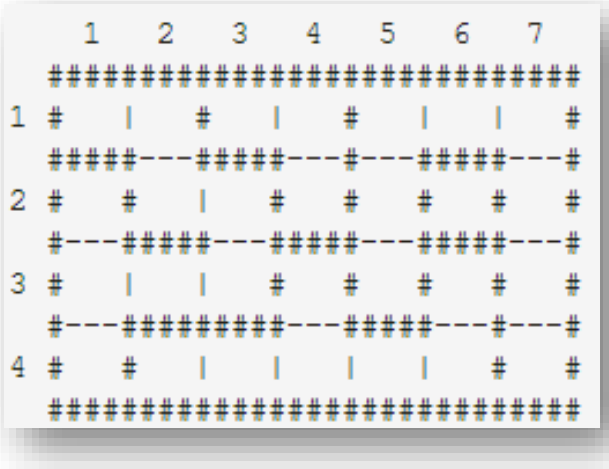
- 对每一个方块，深度优先搜索，从而给这个方块能够到达的所有位置染色
- 最后统计一共用了几种颜色，以及每种颜色的数量
- 例如

  1 1 2 2 3 3 3

  1 1 1 2 3 4 3

  1 1 1 5 3 5 3

  1 5 5 5 5 5 3
- 从而一共有5个房间，最大的房间 (标记1) 占据9个方块



= Wall
| = No Wall
- = No Wall

29

```
#include <iostream>
#include <stack>
#include <cstring>
using namespace std;
int R, C;                 //行列数
int rooms[60][60];
int color[60][60];   //标记房间是否染色过
int maxRoomArea = 0, roomNum = 0;
int roomArea;
void Dfs(int i, int k) {
        if( color[i][k] )
            return;
        ++ roomArea;
        color[i][k] = roomNum;
        if( (rooms[i][k] & 1) == 0 ) Dfs(i, k-1);   //向西
        if( (rooms[i][k] & 2) == 0 ) Dfs(i-1, k);   //向北
        if( (rooms[i][k] & 4) == 0 ) Dfs(i, k+1);  //向东
        if( (rooms[i][k] & 8) == 0 ) Dfs(i+1, k);  //向南
}
```


30

```
int main()  {
        cin >> R >> C;
        for( int i = 1; i <= R;++i)
            for ( int k = 1; k <= C; ++k)
                cin >> rooms[i][k];
        memset(color, 0, sizeof(color));
        for( int i = 1; i <= R; ++i )
            for( int k = 1; k <= C; ++ k ) {
                if( !color[i][k] ) {
                    ++ roomNum;  roomArea = 0;
                    Dfs(i, k);
                    maxRoomArea = max(roomArea, maxRoomArea);
                }
        }
        cout << roomNum << endl;
        cout << maxRoomArea << endl;
}
```


31

```
//解法2: 不用递归, 用栈解决, 程序其他部分不变
void Dfs(int r, int c) {
        struct Room {  int r, c;   Room(int rr, int cc):r(rr), c(cc) { }  };
        stack<Room> stk;
        stk.push(Room(r, c));
        while ( !stk.empty() ) {
                Room rm = stk.top();
                int i = rm.r;   int k = rm.c;
                if( color[i][k] )  stk.pop();
                else {
                        ++ roomArea;
                        color [i][k] = roomNum;
                        if( (rooms[i][k] & 1) == 0 ) stk.push(Room(i, k-1));  //向西
                        if( (rooms[i][k] & 2) == 0 ) stk.push(Room(i-1, k));  //向北
                        if( (rooms[i][k] & 4) == 0 ) stk.push(Room(i, k+1));  //向东
                        if( (rooms[i][k] & 8) == 0 ) stk.push(Room(i+1, k));  //向南
                }
        }
}
```


32

深度优先搜索

寻路问题

寻路问题 ROADS (POJ1724)

N个城市, 编号1到N. 城市间有R条单向道路

每条道路连接两个城市, 有长度和过路费两个属性

Bob只有K元钱, 他想从城市1走到城市N

问最短共需要走多长的路? 如果到不了N, 输出-1

- 2<=N<=100
- 0<=K<=10000
- 1<=R<=10000

每条路的长度 L, 1 <= L <= 100

每条路的过路费T, 0 <= T <= 100

输入:

K
N
R
s_1 e_1 L_1 T_1
s_1 e_2 L_2 T_2
...
s_R e_R L_R T_R
s e是路起点和终点

34

解题思路

从城市 1开始**深度优先遍历**整个图,

找到所有能过到达 N 的走法, 选一个最优的

35

解题思路

从城市 1开始深度优先遍历整个图, 找到所有能过到达 N 的走法, 选一个最优的

优化:

1) 如果**当前已经找到的最优路径长度为L**,

那么在继续搜索的过程中, **总长度已经大于L的走法**, 就可以直接放弃, 不用走到底了

36

解题思路

从城市 1开始深度优先遍历整个图, 找到所有能过到达 N 的走法, 选一个最优的

优化:

2) 用**midL[k][m]**表示:

走到城市k时, 总过路费为m的条件下, 最优路径的长度

若在后续的搜索中, 再次走到k时, 如果总路费恰好为m, 且此时的路径长度已经超过 midL[k][m], 则不必再走下去了

37

```
#include <iostream>
#include <vector>
#include <cstring>
using namespace std;

int K, N, R, S, D, L, T;
struct Road {
    int d, L, t;
};
vector<vector<Road> > cityMap(110); //邻接表
                                    //cityMap[i]是从点i有路连到的城市集合
int minLen = 1 << 30;  //当前找到的最优路径的长度
int totalLen;       //正在走的路径的长度
int totalCost ;    //正在走的路径的花销
int visited[110];  //城市是否已经走过的标记
int minL[110][10100]; //minL[i][j]表示从1到i点的，花销为j的最短路的长度
```


38

```
void Dfs(int s) //从 s开始向N行走
{
        if( s == N ) { //达到终点
                minLen = min(minLen, totalLen);
                return ;
        }
        for( int i = 0 ; i < cityMap[s].size(); ++i ) {
                int d =  cityMap[s][i].d;  //s有路连到d
                if(! visited[d] ) {
                        int cost = totalCost +  cityMap[s][i].t;
                        if( cost > K )
                            continue;
                        if( totalLen +  cityMap[s][i].L >= minLen
                         || totalLen +  cityMap[s][i].L >= minL[d][cost] )
                            continue;
```


39

```
                totalLen += cityMap[s][i].L;
                totalCost +=  cityMap[s][i].t;
                minL[d][cost] = totalLen;
                visited[d] = 1;
                Dfs(d);
                visited[d] = 0;
                totalCost -=  cityMap[s][i].t;
                totalLen -= cityMap[s][i].L;
            }
        }
}
```


40

```
int main()
{
        cin >>K >> N >> R;
        for( int i = 0;i < R; ++ i) {
                int s;
                Road r;
                cin >> s >> r.d >> r.L >> r.t;
                if( s != r.d )
                        cityMap[s].push_back(r);
        }
        for( int i = 0;i < 110; ++i )
                for( int j = 0; j < 10100; ++ j )
                        minL[i][j] = 1 << 30;
        memset(visited,0,sizeof(visited));
        totalLen = 0;
        totalCost = 0;
        visited[1] = 1;
```


41

```
        minLen = 1 << 30;
        Dfs(1);
        if( minLen < (1 << 30))
                cout << minLen << endl;
        else
                cout << "-1" << endl;
}
```


42