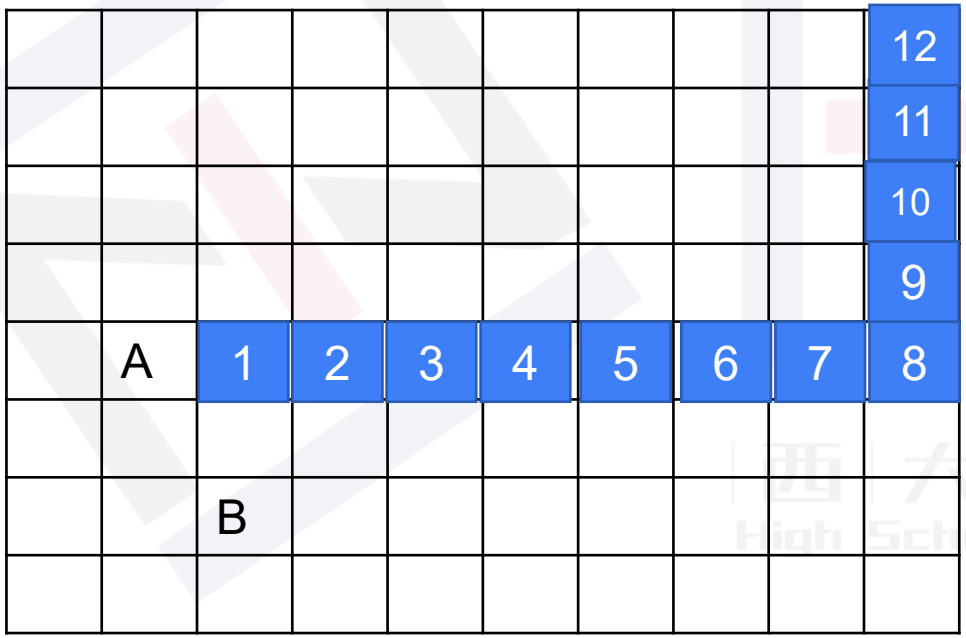

2021
信息学
迭代加深搜索 A* IDA*

复习：一个搜索问题

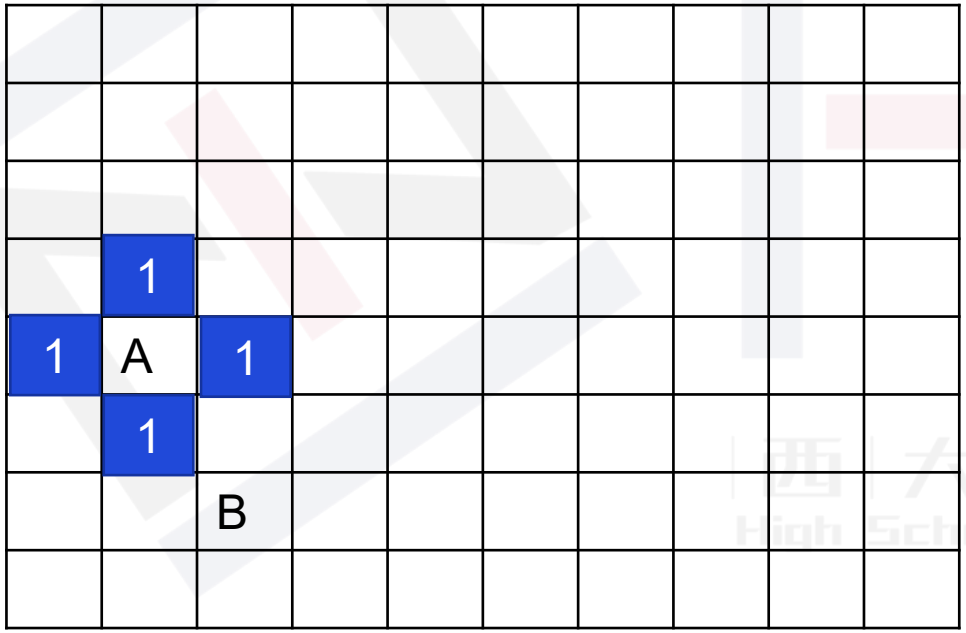


如果从A到B

DFS 可能的搜索路径?

如何优化?

复习：一个搜索问题



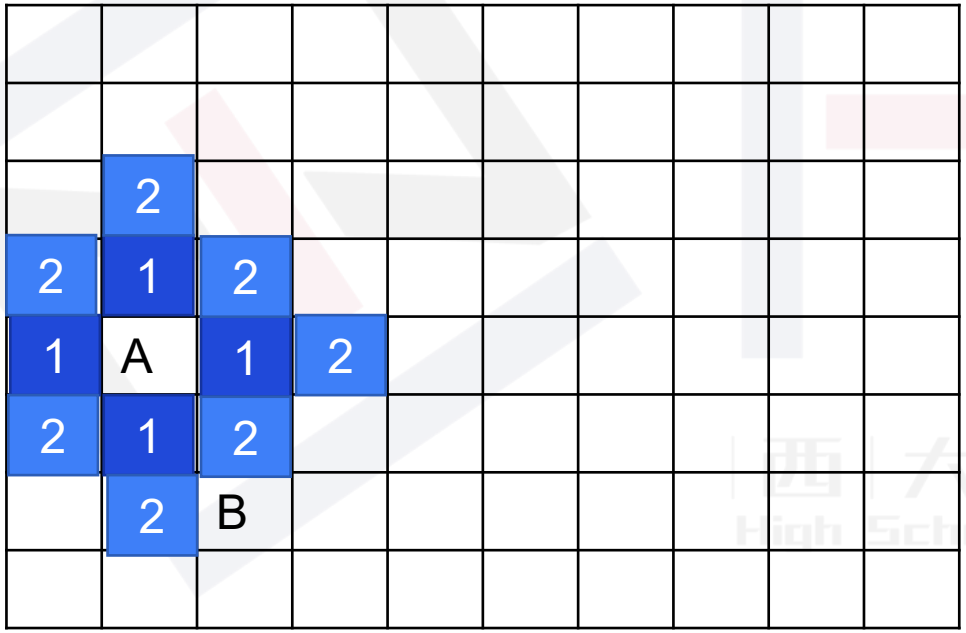
如果从A到B

DFS 可能的搜索路径?

如何优化?

BFS 比较优秀

复习：一个搜索问题



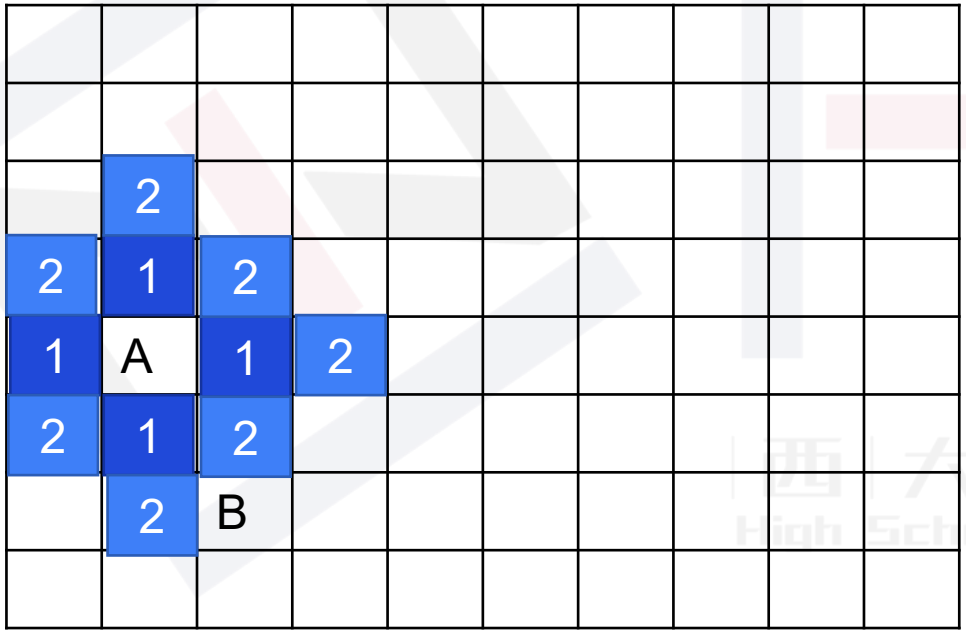
如果从A到B

DFS 可能的搜索路径?

如何优化?

BFS 比较优秀。

复习：一个搜索问题



如果从A到B

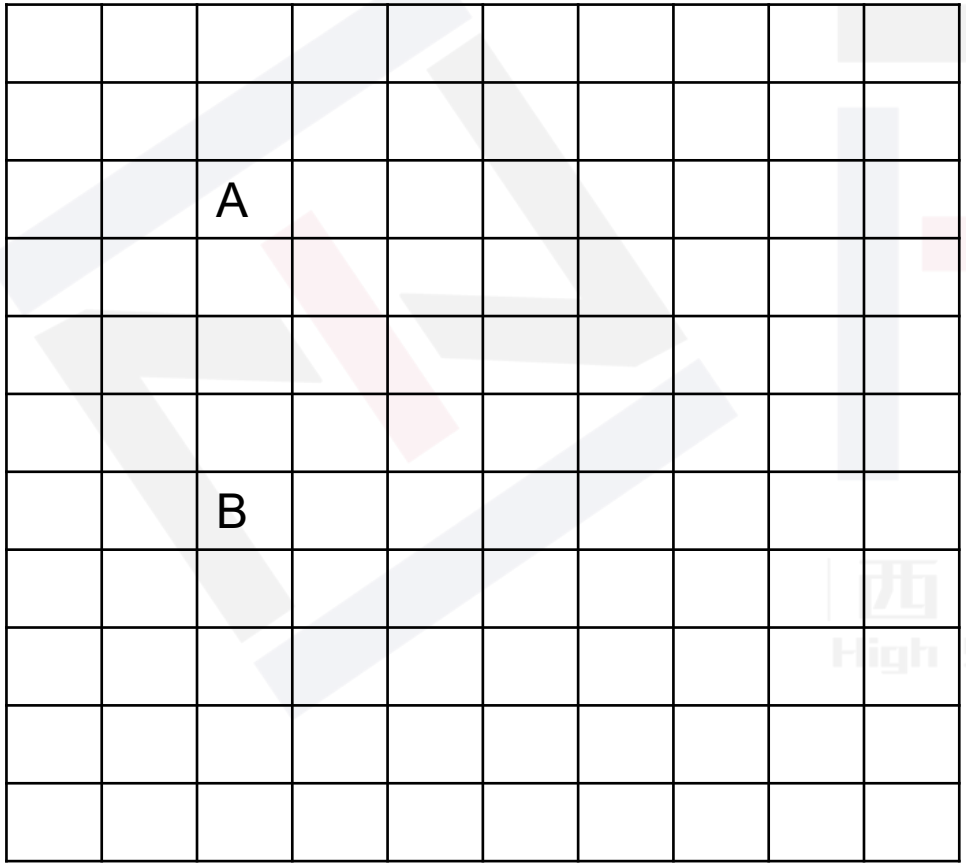
DFS 可能的搜索路径?

如何优化?

BFS 比较优秀。

搜索本质是在多个状态空间组成的图上寻路搜索，找到答案，最终构成一颗搜索树。

新问题：一个搜索问题



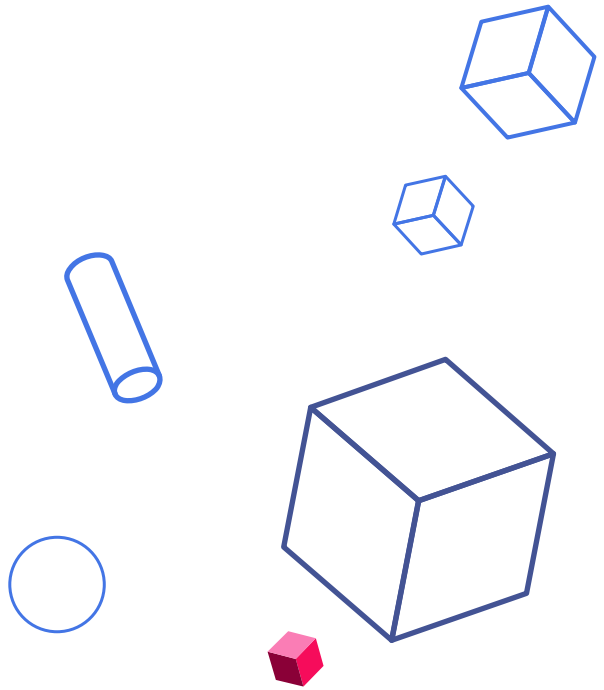
如果搜索空间非常大？但是A和B很近？

分析一下BFS可能会遇到什么情况？

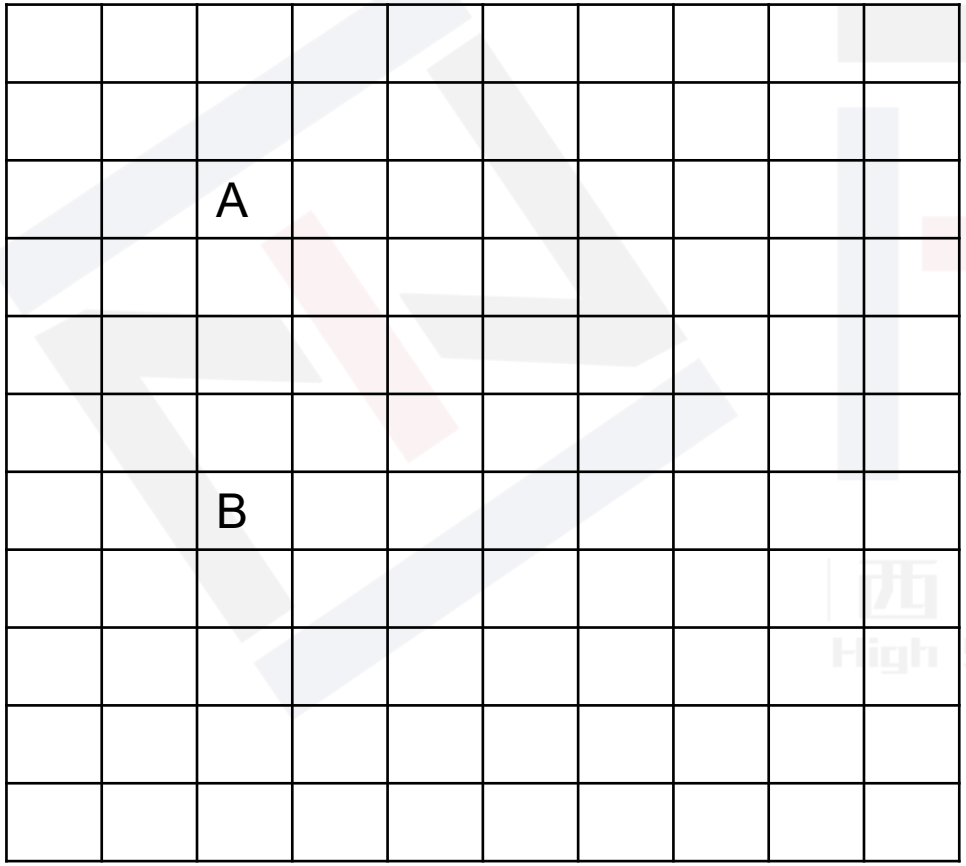
周围的点太多了，在找到B的时候空间炸了

怎么办？

迭代加深搜索



新问题：一个搜索问题

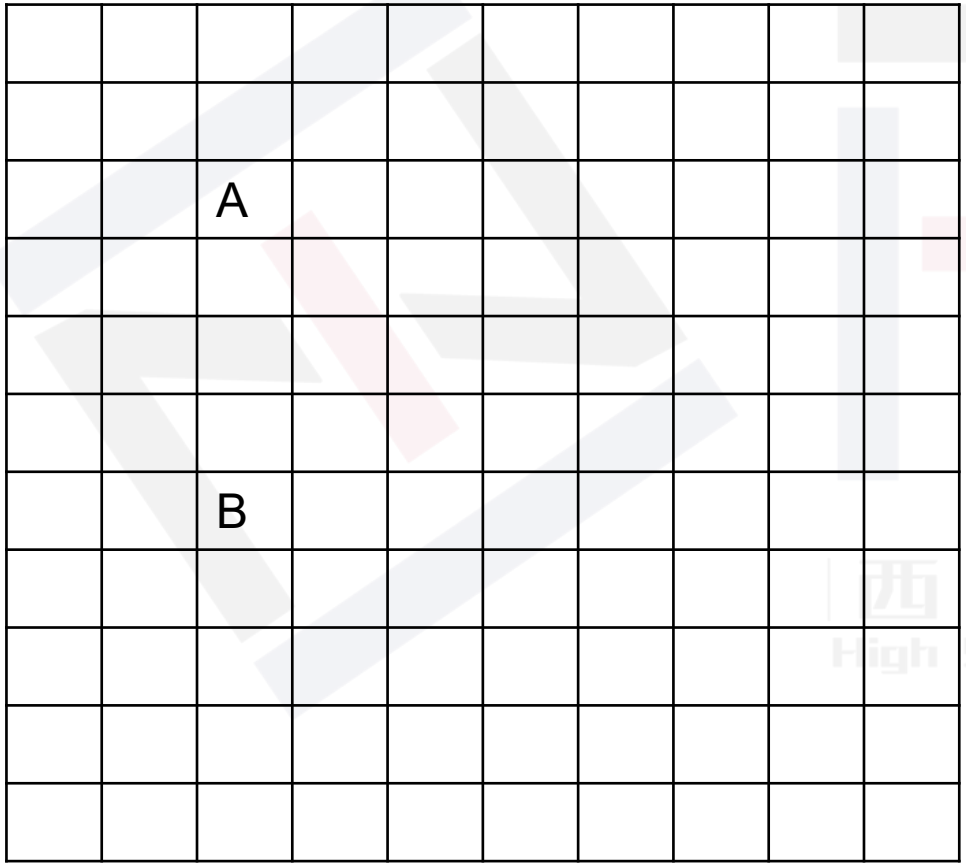


DFS的问题在于一搜停不下来
BFS的问题在于在队列空间开销

给DFS设定一个搜索深度d
如果当前搜完了d还没有找到答案
则d+1 继续搜索。

迭代加深搜索

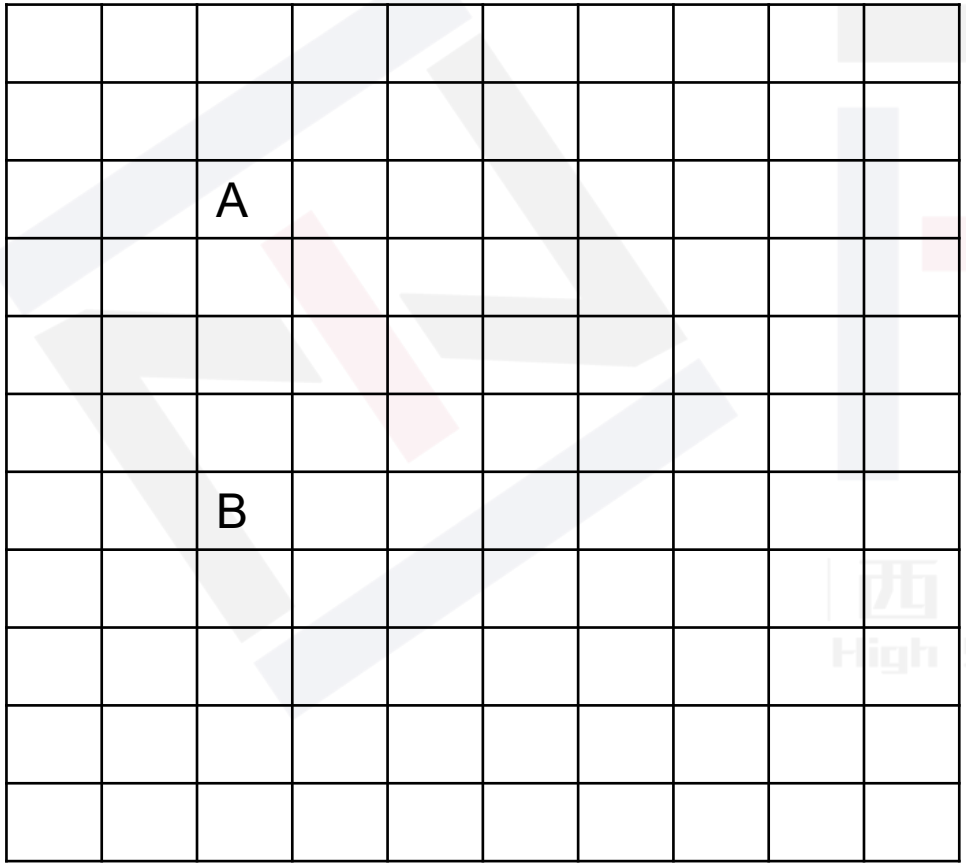
给DFS设定一个搜索深度d
如果当前搜完了d还没有找到答案
则d+1 继续搜索。



画图模拟过程（右）

迭代加深搜索

给DFS设定一个搜索深度d
如果当前搜完了d还没有找到答案
则d+1 继续搜索。



画图模拟过程（右）

适用的前提条件?

搜索空间巨大，但是A和B很近。

伪代码实现

给DFS设定一个搜索深度d
如果当前搜完了d还没有找到答案
则d+1 继续搜索。

```
IDDFS(u,d)
    if d>limit
        return
    else
        for each edge (u,v)
            IDDFS(v,d+1)
  return
```

提示，广搜空间不够用才用迭代，否则直接BFS挺好的。

例：POJ 3134 power calculus

给定一个正整数n，求经过多少次乘法或除法运算可以从x得到x^n，可以使用中间得到的结果

首先，本题显然有解（IDDFS的前提条件，有解，不然_____）

其次，搜索对象确定为选择某一个数进行乘法或除法（理论可行操作空间无限大）

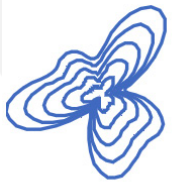
显然，最多做n次就一定能拿到解。所以搜索深度最多到n （满足接近要求）

所以：适用IDDFS 一轮轮搜索即可。

当第一次成功凑出x^n 后，当前深度就是答案。

BFS Dijkstra A*

BFS
过程：略
特点：向每个方向“等价”的搜索



Dijkstra（堆优化）
过程：从点集出发，找一条边，去尝试更新点权（松弛操作）如果该点被更新，则加入到选中的点集中，继续更新其他点。
特点：基于全值的过程性搜索。

A* （读作A star）

Guess

什么样的数据考虑A*？
什么样的数据考虑BFS？

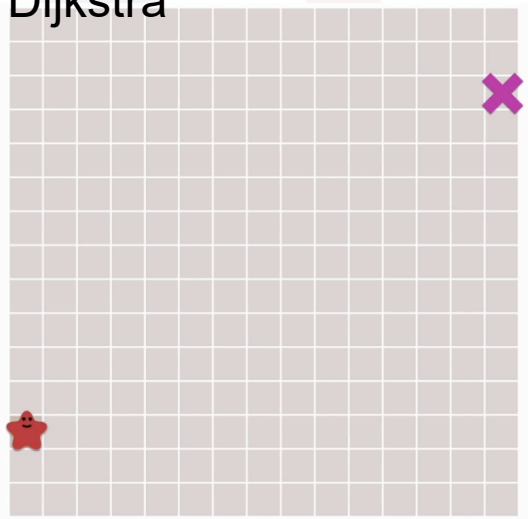
理想的搜索过程应该是怎么样的?

梦中的搜索过程

理想的搜索过程应该是怎么样的?

从起点开始，朝着终点方向搜索！（由终点启发搜索方向）

Dijkstra



贪心：最佳优先搜索算法

实现思路： Greedy Best-First-Search

（注意和广搜的BFS不是同一个东西）

定义一个启发函数,计算A点到终点B的曼哈顿距离

```
int heuristic(N A,N B) {return abs(A.x - B.x) + abs(A.y - B.y); }
```

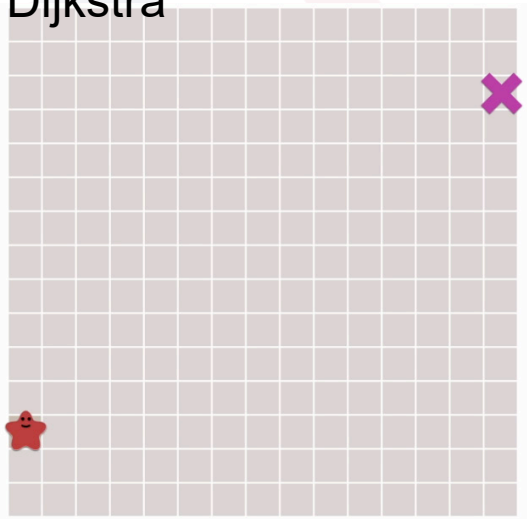
回顾Dji算法，它用把点到起点集的实际距离在优先队列中排序。而这里，我们使用该点到终点的估计距离放在优先队列中维护。

梦中的搜索过程

理想的搜索过程应该是怎么样的?

从起点开始，朝着终点方向搜索！（由终点启发搜索方向）

Dijkstra



贪心：最佳优先搜索算法

伪代码

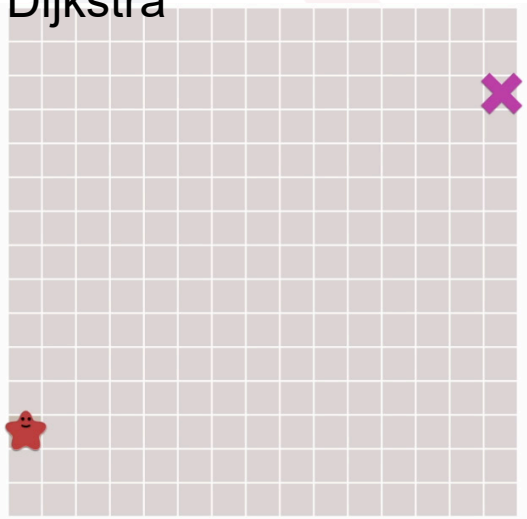
```
Q = PriorityQueue() //优先队列
Q.push(start, 0)
came_from = dict() //记录原点
came_from[start] = None


while not Q.empty():
  current = Q.top()
  Q.pop()


  if current == goal: break

  for next in graph.neighbors(current):
    if next not in came_from:
      p = heuristic(goal, next)
      Q.push(next, p)
      came_from[next] = current
```

梦中的搜索过程

理想的搜索过程应该是怎么样的?

从起点开始，朝着终点方向搜索!

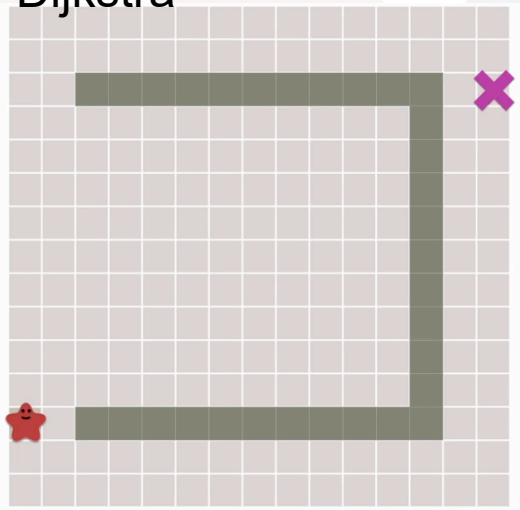
Dijkstra



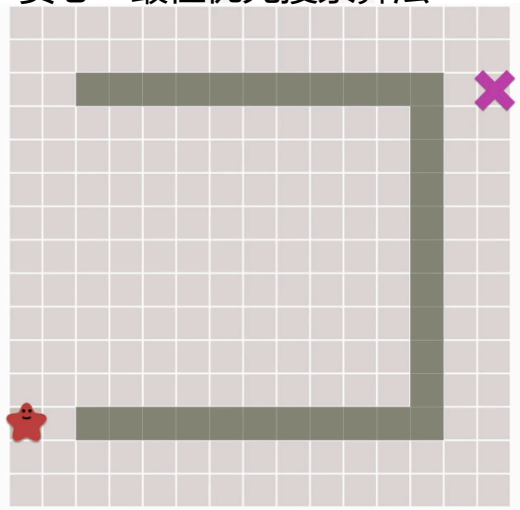
贪心：最佳优先搜索算法

更复杂一点的情况?

Dijkstra



贪心：最佳优先搜索算法



梦中的搜索过程

西南大学附属中学
High School Affiliated to Southwest University

Dji 跑出了正确路径，
但是在显然错误的方向尝试了太多次！

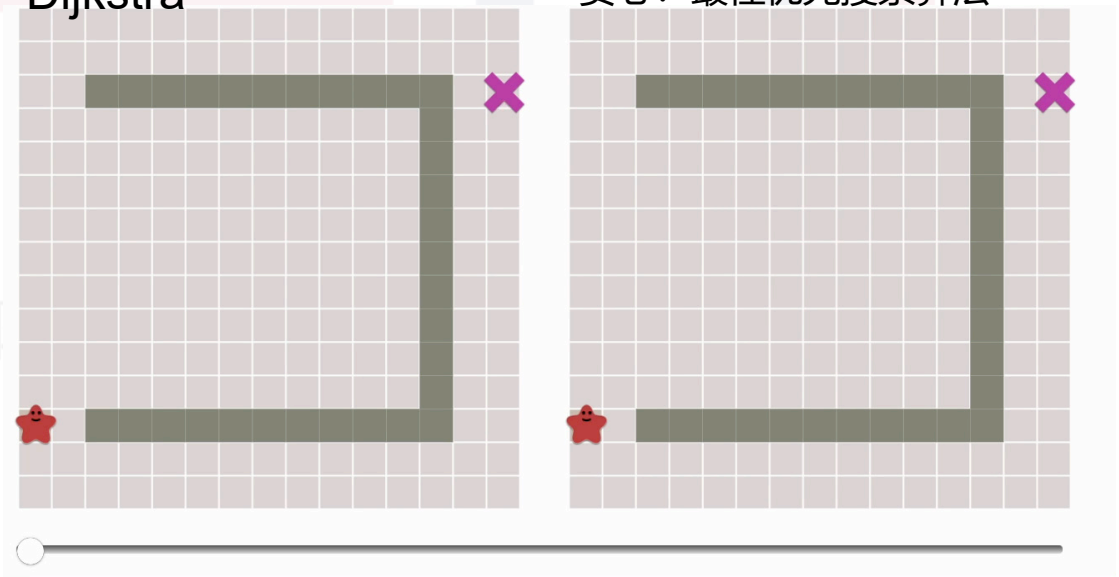
贪心思想搜索跑的快
但是路径不是最短的！

分析

更复杂一点的情况？

Dijkstra

贪心：最佳优先搜索算法

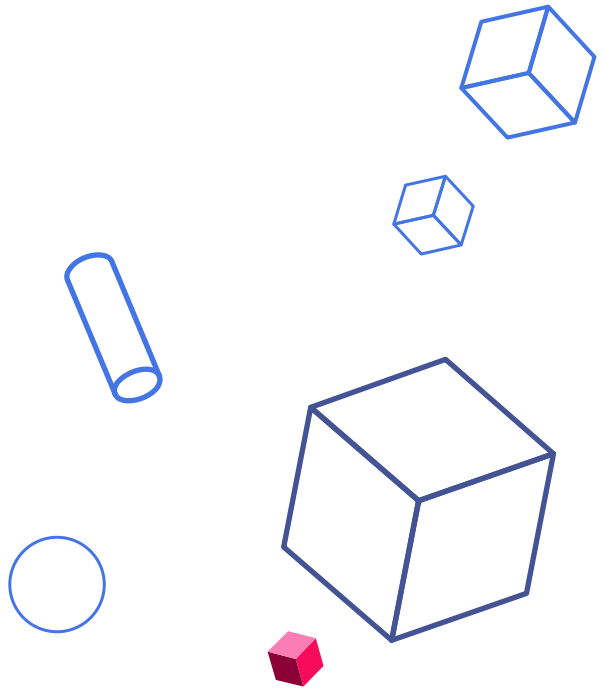


梦中的搜索过程

怎么样跑的快
还能找到正确的最短路？

用写启发函数h的方法，混在Dijkstra中加速
可以看作Dij的一个优化

启发式搜索（heuristic search）
A*算法



感性-理解 A*

- A*搜索算法（英文：A*search algorithm，A*读作 A-star），简称 A*算法，是一种在图形平面上，对于有多个节点的路径求出最低通过成本的算法。它属于图遍历（英文：Graph traversal）和最佳优先搜索算法（英文：Best-first search），亦是 BFS 的改进。
  - ——OI-wiki

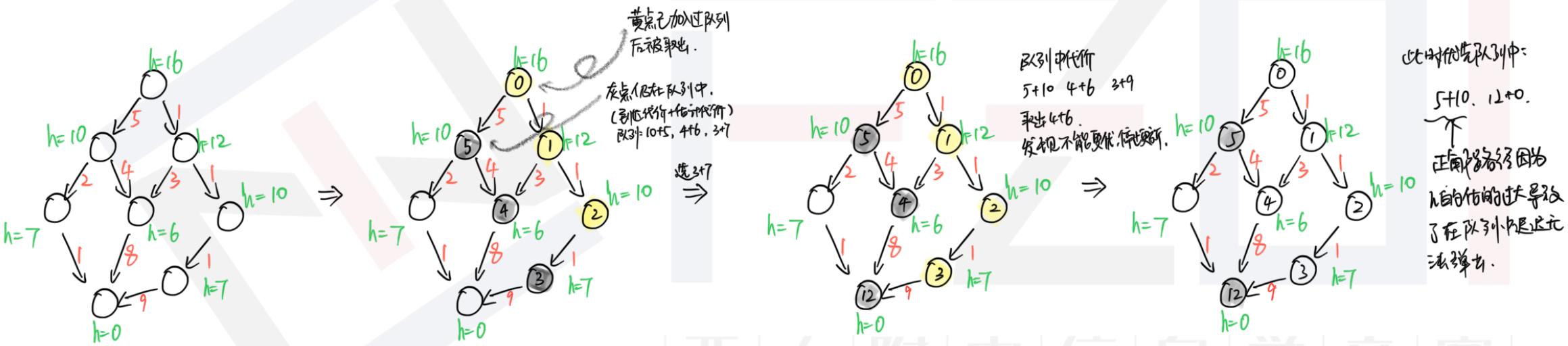
加了估价函数的优先队列BFS称为A*算法

一些细节

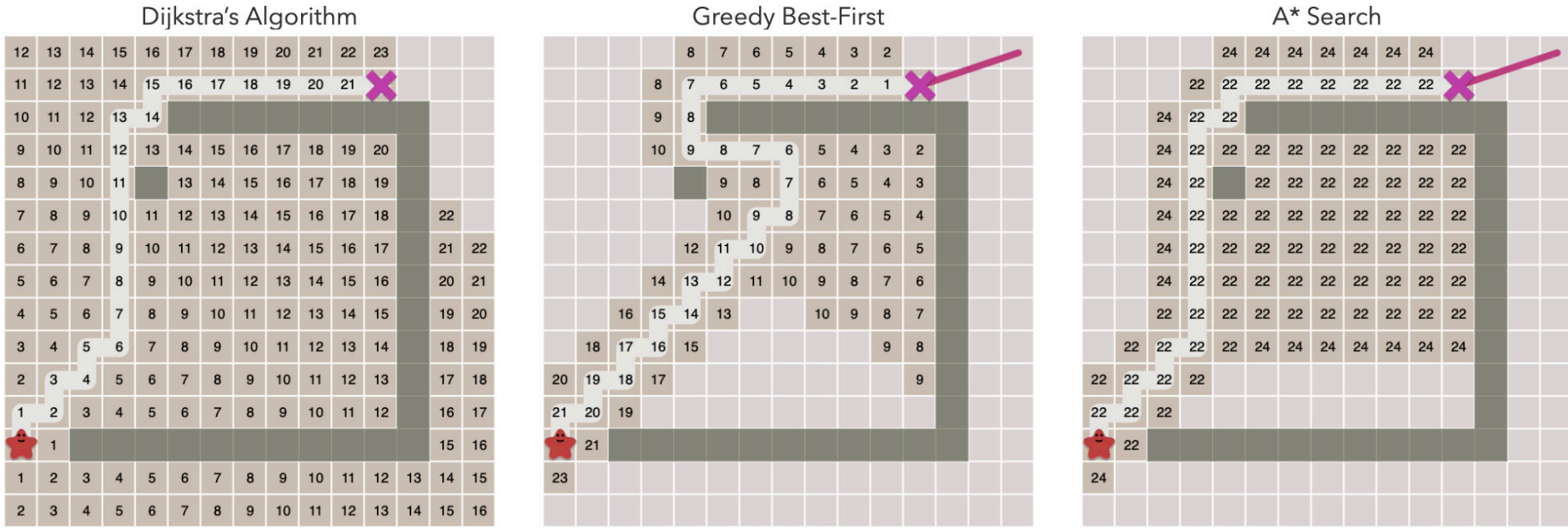
1. 为了对未来产生的代价进行估计，设计一个估价函数，能够预估任意状态到目标状态的未来所需代价的估计值。
2. 使用优先队列时，就需要从堆中取出当前代价+未来估价最小的状态进行扩展。
3. 估价函数h(X)设计十分重要
    - 记：当前状态X到目标所代价估计为h(X)
    - 记：实际代价g(X)
    - 显然：为了确保最优路径 最终能被找到h(x) g(x)之间的的大小关系?

h(x) <= g(x)

为什么h(x) <= G(x)



对比：Dij GBFS A*



A* 具体过程描述

- 1. 把起点加入 open list 。
- 2. 重复如下过程：
- a. 遍历open list ，查找F值最小的节点，把它作为当前要处理的节点，然后移到close list中
- b. 对当前方格的 8 个相邻方格一一进行检查，如果它是不可抵达的或者它在close list中，忽略它。否则，做如下操作：
- □ 如果它不在open list中，把它加入open list，并且把当前方格设置为它的父亲
- □ 如果它已经在open list中，检查这条路径（即经由当前方格到达它那里）是否更近。如果更近，把它的父亲设置为当前方格，并重新计算它的G和F值。如果你的open list是按F值排序的话，改变后你可能需要重新排序。
- c. 遇到下面情况停止搜索：
- □ 把终点加入到了 open list 中，此时路径已经找到了，或者
- □ 查找终点失败，并且open list 是空的，此时没有路径。
- 3. 从终点开始，每个方格沿着父节点移动直至起点，形成路径。

例：K短路

- 给定一张N个点M条边的图，求起点S到终点T得第K短路的长度，路径允许重复经过点或边。
- 1<= S,T <=N <=1e3 0<=M<=1e5 1<=k <= 1e3

怎么做?

回顾Dij求最短路

堆优化Dj算法：

dis[i]表示源点到i的最短路。

每次从堆中取出取出最小的dis[u]，则源点到u的最短路就确定了。同时通过dis[u]更新相邻的点v的dis[v]，更新后入队。

当堆为空时，源点到其他所有点的最短路求解完毕。

时间复杂度为O((N+M)log(N+M))

如果第i次从堆中取出dis[u]，那么对应的dis[u]就是源点到u的第i短路

例：K短路

A* 优化

估价函数满足第K短路中X到T的估计距离h(X)，不大于第K短路中X到T的实际距离g(X)。

设计估价函数h(X)：表示X到T的最短路

求解过程：

（1）预处理各个节点X到终点T的最短路h(X)

建立反向图，求解T为起点的单源最短路。

（2）将二元组(S,0+F[S])放入堆中。

（3）每次从堆中取出dis[x]+h[x]最小的二元组(x,dis[x]+h[x])，

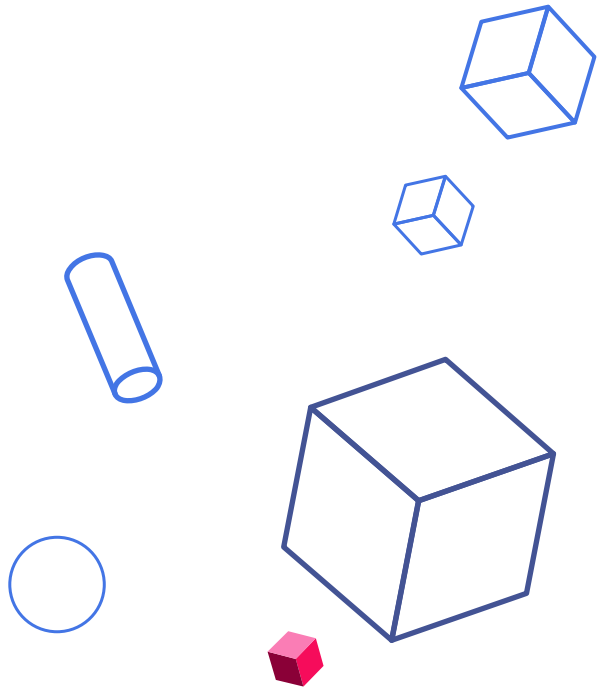
然后将x出发的每条边(x,y)更新，如果y被取出的次数没有达到K次，则将(y,dis[x]+w(u,v)+h[y])入堆。

（4）重复第3步，直到第K次取出包含终点的二元组(T,dis[T]+0)，dis[T]即为答案。

因为估计函数，很多节点访问次数都远小于K，效率更高。

IDA*
Iternative Deepening A*

- 迭代加深 A* 算法



IDA*

迭代加深 A* 算法

IDA*效率比较优秀，并且在实现方面，比A*好实现

还是老问题：
问题空间太大，起终状态接近。
直接使用A*会爆空间（维护一个堆找最小值）

迭代
加深

估价函数+优先队列BFS结合 == A*
估价函数+迭代加深DFS结合 ==IDA*

限制一个搜索深度bound
当搜索代价 + 估计代价 > bound 剪枝
直接bound +1 再来一波

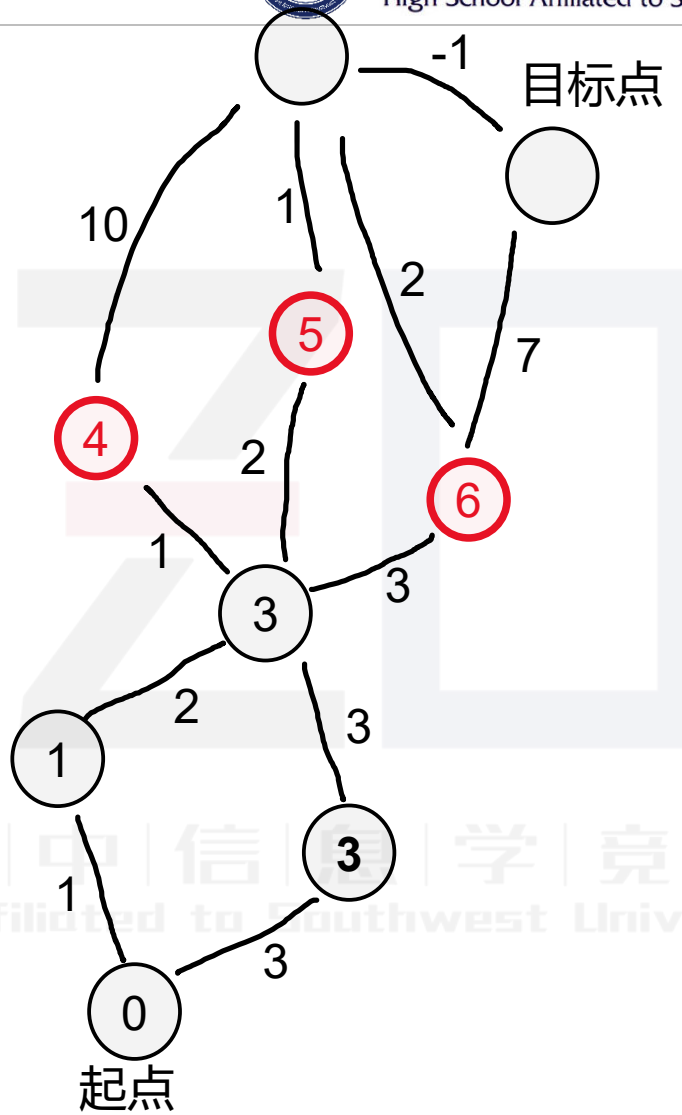
核心思路

继承了A*的关于估价函数h的思想

f(n) = dis(n)+h(n)

if dis(n)+h(n) > bound : return;

f(n)：到达n点搜索下一步方向的决策依据
dis(n)：当前到达n的代价
h(n):从n出发时到达终点的代价



IDA* 伪代码实现


```
void dfs (int dep) {
   int hv = state.h(); //预估当前代价

   if (hv == 0) { flag = 1 ;return; }  //找到解
   if (dep + hv > bound) return ;     //如果代价>bound
   for (int i = 0; i < 4; i++) {
      calc(new_state);  //拓展生成新状态
      dfs(dep+1);  //递归下一层，递归后得到搜索过程中>bound 中最小的 f(n)
      if (flag) return; //找到解后快速结束当前递归过程
   }
}
for (bound = state.h(); !flag;  dfs(++bound));  //循环for bound 直到找到解为止。
```

IDA* 伪代码实现-优化for

//dfs 返回大于 bound 的局面中最小的 f(n), 跳过一些bound 节约时间

```
int dfs (int dep) {
    int hv = state.h(); //预估当前代价

    if (hv == 0) return flag = 1, dep; //找到解
    if (dep + hv > bound) return dep + hv; //如果代价>bound

    int next_bound = 1000; //一个足够大的数

    for (int i = 0; i < 4; i++) {
        calc(new_state);  //拓展生成新状态
        int tmp = dfs(dep+1);  //递归下一层，递归后得到搜索过程中>bound 中最小的 f(n)
        if (flag) return tmp; //找到解后快速结束当前递归过程
        if (tmp < next_bound) next_bound = tmp; //打擂台记录bound
    }
    return next_bound; //优化搜索过程。返回bound
}
for (bound = state.h(); !flag; bound = dfs(0));  //条约循环for bound 直到找到解为止。
```

核心在于h函数的设计！

例1： POJ 3460 Booksort

给定 $n(1 \le n \le 15)$ 本书，编号为 $1 \sim n$。在初始状态下，书是任意排列的。在每一次操作中，可以抽取其中连续的一段，再把这段插入到其他某个位置。我们的目标状态是把书按照 $1 \sim n$ 的顺序依次排列，求最少需要多少次操作。若操作次数 ≥ 5，则直接输出字符串“5 or more”即可。

估计一下暴力搜索的搜索树：
对于n本书，假定从中选i本操作，(1<=i<=n)
那么选取的起点有n-i+1个（选法有n-i+1）
同时，有n-i个位置可插入。
再次，把i本书从后移动向前，等价于从前面部分书籍向后移动，重复计算了1次。
所以每层状态有：

$$\sum_{i-1}^{n}(n-1)(n-i+1) \le \frac{15*14+14*13+\cdots+2*1}{2} = 560$$

一个数学技巧：不等式

例1： POJ 3460 Booksort

给定 $n(1 \le n \le 15)$ 本书，编号为 $1 \sim n$。在初始状态下，书是任意排列的。在每一次操作中，可以抽取其中连续的一段，再把这段插入到其他某个位置。我们的目标状态是把书按照 $1 \sim n$ 的顺序依次排列，求最少需要多少次操作。若操作次数 ≥ 5，则直接输出字符串“5 or more”即可。

题目要求选4次，所以…
复杂度560^4

方法1
考虑双向BFS，从头和尾分别开始搜索，再hash判重（二进制）
复杂度开根号变成560^2

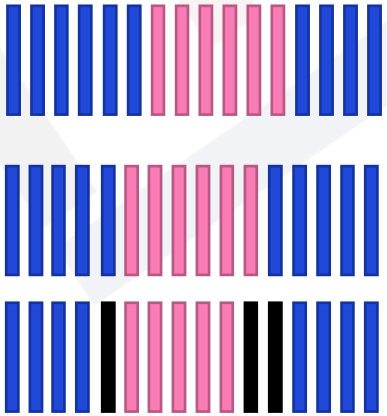
例1： POJ 3460 Booksort

方法2

目标状态要求书籍编号从1～n排列，那么第i本书背后就是第i+1本书，这种情况称为正确情况

假设从当前状态S 调整至目标状态1～n排列

每调整一次，至多可以改变3本书的后继书籍，假设后继书籍的编号错误情况有tot种，



至少需要 ceil (tot/3)（ceil表示向上取整）
那设计一个估价函数 h(x)=ceil (tot/3)
其中h(x)表示在当前x状态下错误的后继总数

基于迭代加深的思想 dep 依次为1～4进行搜索
搜索时枚举选取哪一段移动到后面的哪个位置
估价后移动，
进入函数时，if 代价+h(x) > dep 则回溯

例1： code

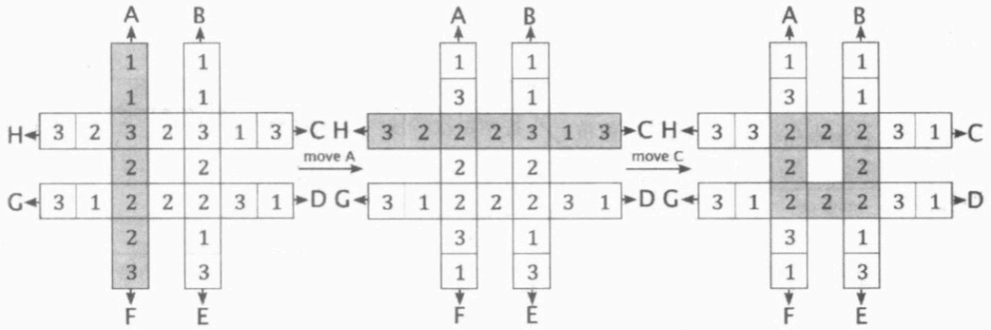
```
#include<iostream>
#include<cmath>
typedef long long ll;
#define N ((ll)2e4+2)
using namespace std;
ll T,n,a[N],b[N],dep;


//计算tot
inline ll f(){
   ll tot=0;
   if(a[n]!=n)tot++;
   for(ll i=1;i<n;i++)
                                  if(a[i+1]!=a[i]+1) tot++;
   return tot;
}
//移动
inline void change(ll x,ll y,ll z){
   ll cnt=0,p=0;
   for(ll i=y+1;i<=z;i++)b[++cnt]=a[i];
   for(ll i=x;i<=y;i++)b[++cnt]=a[i];
   for(ll i=x;i<=z;i++)a[i]=b[++p];
}
```

```
//IDA*
inline bool dfs(ll pos){
  //枚举区间起点终点，后方插入点。
  for(ll i=1;i<n;i++){
    for(ll j=i;j<n;j++){
      for(ll k=j+1;k<=n;k++){
        change(i,j,k);
        if(!f())return 1;//成功找到
        if(pos+ceil(f()/3.0)<=dep){//
          if(dfs(pos+1))return 1;
        }
        change(i,k-j+i-1,k);//回溯
      }
    }
  }
  return 0;
}
int main(){
  std::cin.tie(NULL);
  cin>>T;
  while(T--){
    cin>>n;
    for(ll i=1;i<=n;i++)cin>>a[i];
    if(!f()){cout<<0<<endl;continue;}//本来有序
    bool fg=false;
    for(dep=1;dep<=4;dep++){
                                  //dep全局变量，限制深度，if找到解dfs
返回1
      if(dfs(1)){
        cout<<dep<<endl;
        fg=!fg;
        break;
      }
    }
    if(!fg)cout<<"5 or more"<<endl;
  }
  return 0;
}
```

例2：POJ 2286 Rotation Game

如下图所示，有一个“#”形的棋盘，上面有1，2，3三种数字各8个。给定8种操作，分别为图中的A~H。这些操作会按照图中字母和箭头所指明的方向，把一条长为8的序列循环移动1个单位。例如下图最左边的“#”形棋盘执行操作A后，会变为下图中间的“#”形棋盘，再执行操作C后会变成下图最右边的“#”形棋盘。

给定一个初始状态，请使用最少的操作次数，使“#”形棋盘最中间的8个格子里的数字相同。



用什么方法?

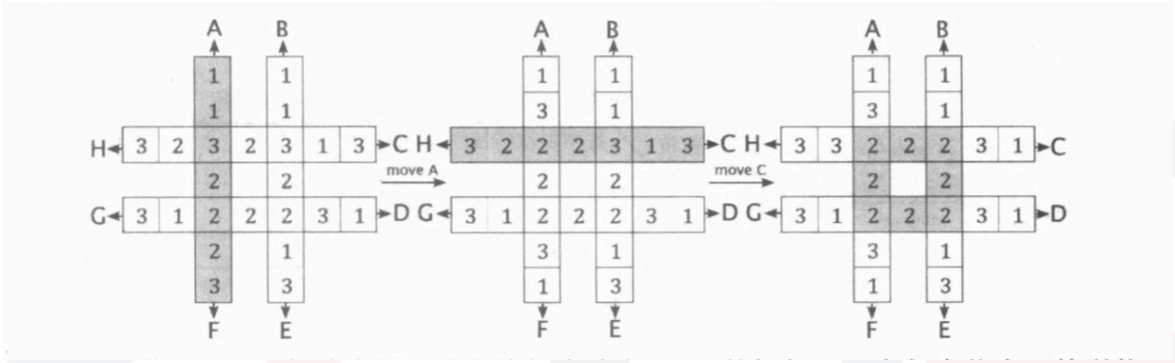
双向BFS？ 问题 不支持BFS拓展 中间状态不明确（搜到什么时候不搜？）

BFS？ 问题 不支持BFS拓展

DFS？ 搜索枚举8个操作（一个显然剪枝：记录之前操作，不能搜回去了）

H函数设计?

例2：估价函数h确定



开脑洞：如果8个数中有最多有k个数是一样的
那么至多需要修改8-k次。
将这个8-k作为估计即可。

然后使用IDA* 迭代加深启发式搜索即可。

例2：code

```
#include<iostream>
#include<cmath>
using namespace std;
int map1[24];
char ans[100];
bool flag=false;
int mov[8][7]={
   {0,2,6,11,15,20,22},   //A
   {1,3,8,12,17,21,23},   //B
   {10,9,8,7,6,5,4},      //C
   {19,18,17,16,15,14,13}, //D
   {23,21,17,12,8,3,1},   //E
   {22,20,15,11,6,2,0},   //F
   {13,14,15,16,17,18,19}, //G
   {4,5,6,7,8,9,10}     //H
};
int hash1[9]={5,4,7,6,1,0,3,2,8};//剪枝，防止回到上一次操作
int center[8]={6,7,8,11,12,15,16,17};
void change(int a){
   int t=map1[mov[a][0]];
   for(int i=0;i<6;i++){
      map1[mov[a][i]]=map1[mov[a][i+1]];
   }
   map1[mov[a][6]]=t;
}


int h(){
   int a[3]={0,0,0};
   for(int i=0;i<8;i++)a[map1[center[i]]-1]++;
   sort(a,a+3);
   return 8-a[2];
}
```

```
int IDAstar(int dep,int bound,int lastMove){
   int t=h();
   if(flag==true) return 1;
   if(dep+t>bound) return 0;
   if(t==0){
      flag=true;
      for(int i=0;i<dep;i++)cout<<ans[i];
      cout<<endl;
      cout<<map1[8]<<endl;
      return 1;
   }
   for(int i=0;i<8;i++){
      if(i!=hash1[lastMove]){
         ans[dep]=i+'A';
         change(i);
         IDAstar(dep+1,bound,i);
         change(hash1[i]);
      }
   }
}




int main(){
   while(1){
      for(int i=0;i<24;i++){
         cin>>map1[i];
         if(map1[0]==0) return 0;
      }


      if(h()==0){
         cout<<"No moves needed"<<endl;
         cout<<map1[8]<<endl;
      }else{
         flag=false;
         for(int i=1;!flag;i++){
            IDAstar(0,i,8);
         }
      }

   }
   return 0;
}
```

参考链接

- 迭代加深搜索
  - https://oi-wiki.org/search/iterative/
  - https://blog.csdn.net/hzaukotete/article/details/81226556
- A*
  - https://www.redblobgames.com/pathfinding/a-star/introduction.html （英文）（主力参考文章）
  - https://www.gamedev.net/reference/articles/article2003.asp
  - https://oi-wiki.org/search/astar/
  - https://zhuanlan.zhihu.com/p/54510444
- IDA*

自学成才

Yi些例题

- BFS
  - 「POJ3322」Bloxorz
  - 「POJ1475」Pushing Boxes
  - 电路维修

- 迭代加深
  - 「POJ2248」Addition Chains
  - 「TYVJ1340」送礼物
  - Problem POJ 2286 - The Rotation Game
  - Bzoj 1085
  - BZOJ1659[Usaco2006 Mar]Lights Out

A*

- 「POJ2449」第 K 短路
- **「POJ1077」Eight**
- 「BZOJ1085」骑士精神