POJ3984

题目描述（POJ3984）：用一个二维数组表示一个迷宫，其中 1 表示墙壁，0 表示可以走的路，只能横着走或竖着走，不能斜着走，编写程序，找出从左上角到右下角的最短路线。

```
int maze[5][5] = {
0, 1, 0, 0, 0,
0, 1, 0, 1, 0,
0, 0, 0, 0, 0,
0, 1, 1, 1, 0,
0, 0, 0, 1, 0,
};
```

输入：一个 5×5 的二维数组，表示一个迷宫。数据保证有唯一解。

输出：从左上角到右下角的最短路径，格式如以下输出样例所示。

输入样例	输出样例
0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0	(0, 0) (1, 0) (2, 0) (2, 1) (2, 2) (2, 3) (2, 4) (3, 4) (4, 4)

题解：本题为典型的迷宫问题，可以使用广度优先搜索解决。定义方向数组 dir[4][2]= {{1,0},{-1,0},{0,1},{0,-1}}，定义前驱数组 pre[][]记录经过的节点。

1. 算法设计

（1）定义一个队列，将起点(0, 0)入队，标记已走过。

（2）如果队列不空，则队头出队。

（3）如果队头正好是目标(4, 4)，则退出。

（4）沿着 4 个方向搜索，如果该节点未出边界、未走过且不是墙壁，则标记走过并入队，用前驱数组记录该节点。

（5）转向步骤 2。

（6）根据前驱数组输出从起点到终点的最短路径。

2. 算法实现

```
void bfs(){
   queue<node> que;
   node st;
   st.x=st.y=0;
   que.push(st);
   vis[0][0]=1;
   while(!que.empty()){
      node now=que.front();
      que.pop();
      if(now.x==4&&now.y==4)
         return;
      for(int i=0;i<4;i++){
         node next;
         next.x=now.x+dir[i][0];
         next.y=now.y+dir[i][1];
         if(next.x>=0&&next.x<5&&next.y>=0&&next.y<5&&!mp[next.x][next.y]&&
!vis[next.x][next.y]){
            vis[next.x][next.y]=1;
            que.push(next);
            pre[next.x][next.y]=now;
         }
      }
   }
}

void print(node cur){//输出路径
   if(cur.x==0&&cur.y==0){
      printf("(0, 0)\n");
      return;
   }
   print(pre[cur.x][cur.y]);//递归
   printf("(%d, %d)\n",cur.x,cur.y);
}
```

POJ3635

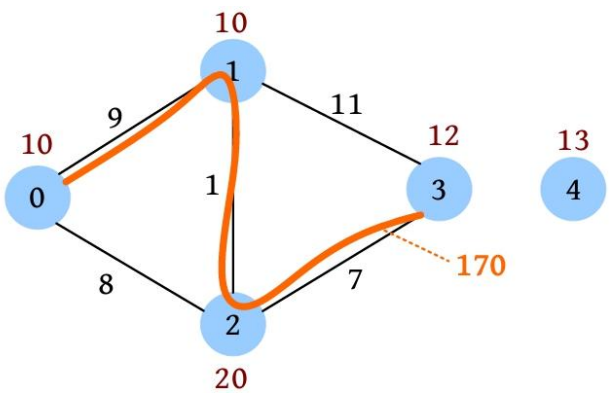
题目描述（POJ3635）：城市之间的油价是不一样的，编写程序，寻找最便宜的城市间旅行方式。在旅行途中可以加满油箱。假设汽车每单位距离使用一单位燃料，从一个空油箱开始。

输入：输入的第 1 行包含 n（1≤n≤1000）和 m（0≤m≤10 000），表示城市和道路的数量。下一行包含 n 个整数 p_i（1≤p_i≤100），其中 p_i 表示第 i 个城市的燃油价格。接下来的 m 行，每行都包含 3 个整数 u、v（0≤u, v<n）和 d（1≤d≤100），表示在 u 和 v 之间有一条路，长度为 d。接下来一行是查询数 q（1≤q≤100）。再接下来的 q 行，每行都包含 3 个整数 c（1≤c≤100）、s 和 e，其中 c 是车辆的油箱容量，s 是起点城市，e 是终点城市。

输出：对于每个查询，都输出给定容量的汽车从 s 到 e 的最便宜旅程的价格，如果无法从 s 到 e，则输出“impossible”。

输入样例	输出样例
5 5 10 10 20 12 13 0 1 9 0 2 8 1 2 1 1 3 11 2 3 7 2 10 0 3 20 1 4	170 impossible

题解：本题为加油站加油问题。给定 n 个节点、m 条边，每走 1 单位的路径都会花费 1 单位的油量，并且不同的加油站价格是不同的。现在有一些询问，每一个询问都包括起点、终点及油箱的容量，求从起点到达终点所需的最少花费。可以采用优先队列分支限界法搜索。



涉及两个维度的图最短路径，一个是费用，一个是地点。可以把当前节点对应的油量抽象成多个节点（例如在位置 0 有 1 升油是一个节点，在位置 0 有 2 升油是一个节点），把费用看作边，那么最少费用就可以类似 Dijsktra 算法那样不断地加入节点。于是得到一个扩展节点的策略：每次都加 1 升油；如果依靠加的油足够走到下一个节点，就走到下一个节点（减去路上消耗的油，花费不变）；在广度优先搜索中将所有可扩展的节点都加入优先队列中，如果到达终点，则返回花费。

1. 算法设计

（1）定义一个优先队列，将起点及当前油量、花费作为一个节点(st,0,0)入队。

（2）如果队列不空，则队头(u,vol,cost)出队，并标记该节点油量已扩展，vis[u][vol]=1。

（3）如果 u 正好是目标 ed，则返回花费 cost。

（4）如果当前油量小于油箱容量，且 u 的油量 vol+1 未扩展，则将该节点(u,vol+1, cost+price[u])入队。

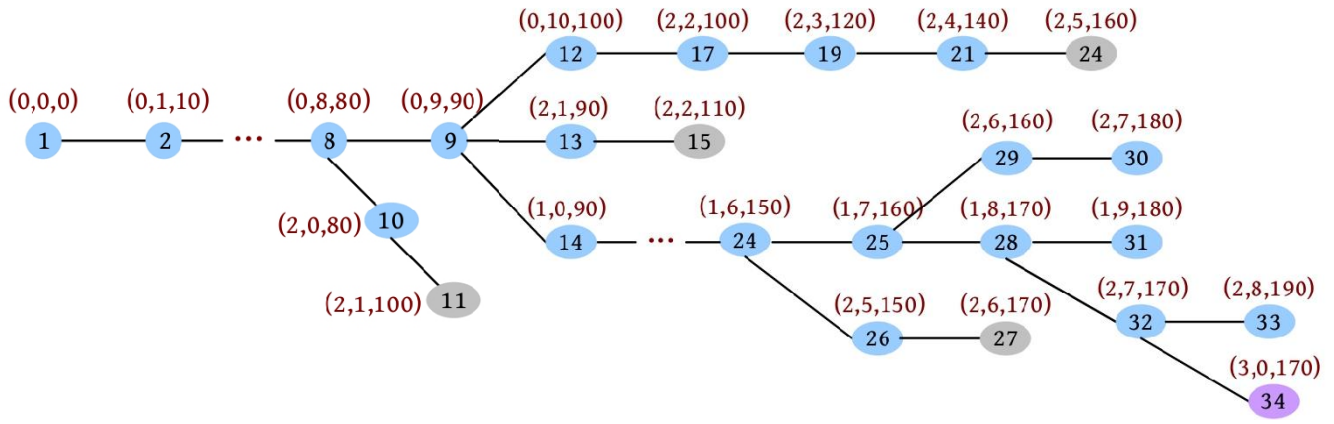
（5）检查 u 所有邻接点 v，如果当前油量大于或等于边权 w，且 v 节点的油量 vol-w 未扩展，则将该节点(v,vol-w,cost) 入队。

（6）转向步骤 2。

2. 算法优化

用一个数组 dp[i][j]表示在节点 i、当前油量为 j 时的最小花费。在当前节点及油量对应的花费更小时才生成节点，生成的节点会少很多，但由于系统数据量不大，运行时间差不多。

采用优化后的算法生成节点的过程如下图所示。



3. 算法实现

```
struct node{//节点类型
   int u,vol,cost;//节点、当前油量、花费
   node(int u_,int vol_,int cost_){u=u_,vol=vol_,cost=cost_;}
   bool operator < (const node &a) const{
      return cost>a.cost;//最小值优先
   }
};

int bfs(){
   memset(vis,0,sizeof(vis));
   priority_queue<node>Q;
   Q.push(node(st,0,0));
   while(!Q.empty()){
      node cur=Q.top();
      Q.pop();
      int u=cur.u,vol=cur.vol,cost=cur.cost;
      vis[u][vol]=1;
      if(u==ed) return cost;
      if(vol<V&&!vis[u][vol+1])//加1单位的油
         Q.push(node(u,vol+1,cost+price[u]));
      for(int i=head[u];~i;i=edge[i].next){
         int v=edge[i].v,w=edge[i].w;
         if(vol>=w&&!vis[v][vol-w])
            Q.push(node(v,vol-w,cost));
      }
   }
   return -1;
}

int bfs(){//算法优化
   memset(vis,0,sizeof(vis));
   memset(dp,0x3f,sizeof(dp));
   priority_queue<node>Q;
   Q.push(node(st,0,0));
   dp[st][0]=0;
   while(!Q.empty()){
      node cur=Q.top();
      Q.pop();
      int u=cur.u,vol=cur.vol,cost=cur.cost;
      vis[u][vol]=1;
      if(u==ed) return cost;
      if(vol<V&&!vis[u][vol+1]&&dp[u][vol]+price[u]<dp[u][vol+1]){
         dp[u][vol+1]=dp[u][vol]+price[u];
         Q.push(node(u,vol+1,cost+price[u]));
      }
      for(int i=head[u];~i;i=edge[i].next){
         int v=edge[i].v,w=edge[i].w;
         if(vol>=w&&!vis[v][vol-w]&&cost<dp[v][vol-w]){
            dp[v][vol-w]=cost;
            Q.push(node(v,vol-w,cost));
         }
      }
   }
   return -1;
}
```