COMP 2012H
Fall 2015

# Midterm Solutions

1. (a)
```
bool canSkiTowards(int map[MAX_ROW][MAX_COL], int row, int col,
                   Direction dir)
{
  switch(dir)
    {
    case NORTH:
      return row-1 >= 0 && map[row-1][col] != -1 &&
             map[row-1][col] <= map[row][col];
    case SOUTH:
      return row+1 < MAX_ROW && map[row+1][col] != -1 &&
             map[row+1][col] <= map[row][col];
    case EAST:
      return row+1 < MAX_COL && map[row][col+1] != -1 &&
             map[row][col+1] <= map[row][col];
    case WEST:
      return row-1 >= 0 && map[row][col-1] != -1 &&
             map[row][col-1] <= map[row][col];
    }
}
```

(b)
```
bool isReachableRecur(int map[MAX_ROW][MAX_COL],
     bool visited[MAX_ROW][MAX_COL], int i, int j, int a, int b)
{
  if( i == a && j == b )
    {
      return true;
    }

  visited[i][j] = true;
  if(canSkiTowards(map,i,j,NORTH))
    {
      if(!visited[i-1][j] && isReachableRecur(map,visited, i-1, j, a, b))
       {
         return true;
       }
    }

  if(canSkiTowards(map,i,j,SOUTH))
    {
      if(!visited[i+1][j] && isReachableRecur(map,visited, i+1, j, a, b))
       {
         return true;
       }
    }

  if(canSkiTowards(map,i,j,EAST))
    {
      if(!visited[i][j+1] && isReachableRecur(map,visited, i, j+1, a, b))
        {
          return true;
        }
    }

  if(canSkiTowards(map,i,j,WEST))
    {
      if(!visited[i][j-1] && isReachableRecur(map,visited, i, j-1, a, b))
        {
          return true;
        }
    }
```

```
  return false;
}

bool isReachable(int map[MAX_ROW][MAX_COL], int i, int j, int a, int b)
{
  bool visited[MAX_ROW][MAX_COL] = {false}; // set the first entry as false,
                                    // and the rest as default, i.e., false

  return isReachableRecur(map, visited, i, j, a, b);
}
```

(c) Method 1 (recursive solution):

```
int lowestInKSteps(int map[MAX_ROW][MAX_COL], int i, int j, int k)
{
  if( k == 0)
    return map[i][j];

  int lowestHeight = map[i][j];

  for(int dir = NORTH; dir <= WEST; dir++)
    {
      if(canSkiTowards(map, i, j, (Direction)dir))
        {
          int newHeight = lowestInKSteps(map, i-(dir==NORTH)+(dir==SOUTH),
                          j+(dir==EAST)-(dir==WEST), k-1);
          if(newHeight < lowestHeight)
            lowestHeight = newHeight;
        }
    }

  return lowestHeight;
}
```

Method2 (Making use of Part 1b):

```
int lowestInKRounds2(int map[MAX_ROW][MAX_COL], int i, int j, int k)
{
  int lowestHeight = map[i][j];
  for(int r = 0; r < MAX_ROW; r++)
    {
      for(int c = 0; c < MAX_COL; c++)
        {
          int diff_r = r - i;
          int diff_c = c - j;
          if(diff_r < 0)
             diff_r = -diff_r;
          if(diff_c < 0)
             diff_c = -diff_c;

          if((diff_r + diff_c <= k) && isReachable(map, i, j, r, c))
            {
              if(map[r][c] < lowestHeight)
                lowestHeight = map[r][c];
            }
        }
    }
  return lowestHeight;
}
```

2. (a)
```
int index( char ch ){
  return (ch - 'a');
}
```

(b)
```
void Insert(Trie *root, const char* s)
{
  Trie *p=root;
  while(*s!='\0')
    {
      if(p->next[index(*s)]==NULL)
        {
          p->next[index(*s)]=createNode();
        }
      p=p->next[index(*s)];
      ++s;
    }
  p->exist=true;
}
```

(c)
```
void Read(char* filename, Trie *root)
{
  std::ifstream fin;
  char str[MAX_LENGTH+1]; //the word in the line including trailing '\0'

  fin.open(filename,std::ifstream::in);
  if(fin.fail())
    {
      printf("Open failure...\n");
      exit(1);
    }
  else
    {
      while(!fin.eof())
        {
          fin.getline(str,MAX_LENGTH,'\n');
          Insert(root,str);
        }
      fin.close();
    }
}
```

(d)
```
bool Search(Trie* root, const char* s)
{
  Trie *p=root;
  while(*s!='\0')
    {
      p=p->next[index(*s)];
      if(p==NULL) return 0;
      ++s;
    }
  return p->exist;

  /* another recursive method
     if (!root)
     return false;
     else if (!*s)
     return root->exist;
     else
     return Search(root->next[index(*s)],s+1);
  */
}
```

(e)
```
void PrintWords(Trie *root,char *buffer, int level)
{
  if (root == NULL)  // no character
    return;
  if (root->exist)  // word completion
    {
      buffer[level]='\0';
      printf("%s\n", buffer);
    }
  for (int i = 0; i < MAX_NCHAR; ++i)
    {
      buffer[level] = 'a' + i;  // assume a character is formed
      PrintWords(root->next[i], buffer, ++level); // move one level down
      level--; //backtrack; move one level up to print out other words
    }
}

void Print (Trie *root){
  char buffer[MAX_LENGTH+1];
  int pos = 0;  // starting level

  PrintWords(root,buffer,pos);  // print words starting from the root
}
```