Discrete event simulation: An application of priority queues.

Time step simulation

Stacks, Queues, Priority Queues

Mix-and-Match

Kinds of Variables and which to use in Prolog

CSI 310: Lecture 25
CS Dept. Conf. Room 11-98
1:00PM
Thur., Apr. 29, 2004
Justin Staubach
Undergraduate Honors Seminar
Utility
Integrating Integrity Checks Into the Dump/Restore
New for CIS1310: When a function recurses, multiple activation records for it are allocated in the execution stack, so multiple variables exist (are alive) with the

when the corresponding function activation returns.
The lifetime ends when execution leaves the containing block; in particular,
taught first in CIS201. Most common and elementary.

\{ ... \};
int HereIsanautomaticharry[1039];
main() { int Imautomatichall; int Anotherauto; Maze SoamiToo;

Automatic:

In C++, there are 3 basic kinds of lifetimes (which came from C):

Each object (synonym variable) has a lifetime.

Each object has a compile time (static) type, which determines which operations
determine what operations

dynamic)

Each object has a run time (dynamic)

Obj ect = Variable = Instance

In C++ terminology, these words are synonyms:

University at Albany Computer Science Dept.
automatic variable.

A C++ constructor is called when execution flows into the definition of an
same variable name.
we use library functions malloc() and free() instead.

This lifetime is controlled by explicitly coded new and delete operations. (In C,

The 2nd basic kind of lifetime is DYNAMIC.
have LOCAL SCOPE except those declared externally.

Another confusing feature of C is static INSIDE a function body makes the

the variable's name have the SCOPE instead of GLOBAL SCOPE.

Another favorite feature of classical programs, like Linux Torvalds, is the

Classical/C style code often has static variables shared by several functions.

be static.

The 3rd basic kind of lifetime is STATIC.
instance) containing them. So, all 3 kinds are possible.

All other data members have the same lifetime as the particular object (i.e.,

(only one instance for that class).

static data members have \texttt{STATIC} lifetime. They are called \texttt{class variables}.

What about data members of classes/structs?
void printPath(node *node)
{
    if (node == 0 || node->size == 0 || node->size < size)
        return;

    if (node->isEmpty_hexagon) // inside the maze.
        postReturn true iff
    int size; // Invariant: Maze[0...size-1][0...size-1] hold our maze
    int Maze[64][64];

    private:
}

class Maze {
    "include "node.h"
    #define MazeIncluded
    #define MazeIncluded
    #defere MazeIncluded
    #include MazeIncluded

    "Maze.h"

    Implementation is functionality and access its private data members.
    Automatic variables belonging to main( ), and use function members to
    The choice Main/Savitch illustrated most was: Make the key class instance be an
    This is a software design choice.

    So, what lifetime should I choose for the Maze?
```java
void print() {
    // the assignment
    // by the assignment
}

Maze() {
    postcondition: The size and Maze contents have been read
    // postcondition: The size and Maze contents have been read
    // from stdin and stored in these data members as specified
    // from stdin and stored in these data members as specified
    // or paths is printed too, after all the paths, as specified by
    // If this Linked List only contains 1 hexagon position, the number
    // of paths is printed too, after all the paths, as specified by
    // path stored in the Linked List whose first node is addressed by
    // this algorithm (0, 0) to (size-1, size-1) among those that begin with the
    // void solve(Tnode *)
    } // post: What is printed is all simple paths
```
{ int main()
    
    #include "Maze.h"

    Maze MyMaze(); // reads in and constructs the maze.
    MazeNode.print();
    MazeNode.print();
    MazeNode.print();
    MazeNode.print();

    int pstart = new MazeNode;
    pstart->i = 0;
    pstart->j = 0;
    pstart->link = NULL;

    MazeNode.solve(pstart); // prints all solutions and their count.

    return 0;
}
see below...

Answer: The execution stack which contains the automatic variables named g,
should then try an UPPER-RIGHT, then LEFT, then RIGHT, ... all six?

How does the computer figure out that after trying an UPPER-LEFT step, it

Where is the stack of trial decisions?

The search algorithm follows the backtracking strategy of the n queens solution

...
Be Smart: Let's Mix and Match!
class SList {
    Item A[N];
    ...
    static const int N = 5;
    SList MySL;
}

Element-by-element dynamic allocation

class DList {
    struct Node {
        Item data;
        Node *link;
    };
    Node *Head;
    ...
    class DList<T>
    {
        Node *Head;
        Item *next;
        ...
    }
    DList<int> MyDL;
    MyDL.insert(15);
    MyDL.insert(72);
    MyDL[72]
    MyDL[15]
}

Static sized allocation of a List.


```c
struct Node {
  Item data;
  Node *link;
};

template <class Item>
CACHED dynamic allocation
Node *array;
int capacity;
int used;

class CList {
  CList<int> MyCL(5);
  MyCL.insert(15);
  MyCL.insert(72);
};

struct Node {
  Item data;
};

text Clist

class CList
  
  template <class Item>
  CACHED dynamic allocation

```
(Linked) List of Statically SIZED Lists:

NULL

5
4
3
2
1
0

3
6
9
8
7

University at Albany Computer Science Dept
buffers would be a better data structure choice. It is illustrated next.

For such new applications, one-by-one dynamic allocation of dynamically sized small packets occur in telnet/ssh protocol traffic. Such a maximum size buffer for a packet whose data size is only 1-2 bytes. (Such much larger, like several megabytes. So, it is very wasteful of memory to allocate future network hardware and protocols may have packet size limits that are

number generated by software and yet to be transmitted. The predictable is number of packets received and yet to be processed, and the predictable is number of packets sent and yet to be received. However, what is not limited by a fixed amount, typically 1500 bytes or so. However, what is not network cards send and receive packets from the network media whose sizes are

computation running a network interface.

The preceding slide illustrates a data structure like that in every modern
The value of expression `A[I]` is the address of row `I`.

```
int A[5][7];
//static sized 2-dim
//array of ints.
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```

```
int A[5][7];
```
8
Row 1: 4 5 6
Row 0: 1 2 3

{ {13, 14, 15}
{10, 11, 12}
{7, 8, 9 }
{4, 5, 6 }
{3, 2, 1 } } = [N_COLS][N_ROWS] 
int main() 
{ 
} ( )
*/

{ count >> [N] [ ] 
  for (int j = 0; j < N_COLS; j++) 
  { count >> ( ) 
  } ( ) 
  for (int i = 0; i < N_ROWS; i++) 
  { count >> ( ) 
  #include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>
(Linked) List of Dynamically Sized Lists:
Here is a "ragged array"; in particular, an array of pointers to C-strings:

variables: main(int argc, char *argv[]) 
structure partly contained in auto parameter

Then, the above diagram describes the data

Suppose you run your program named

proj7 inputfile.dat 35

(under Unix) using the command line below:

Here is a "ragged array"; in particular, an array of pointers to C-strings:
are needed for the 11 NON-ZERO entries.
Instead of 25 cells, only 11 nodes
of this SPARSE 5x5 MATRIX.

Above is the linked list representation
of this entry's column index.
The entry's value is also stored in
its node.

Here is this entry's row address.
be useful. But the structures themselves probably will continue to change over the decades. The programming languages you use to express these structures are likely to vary because features like pointers, structures, and dynamically allocated language because data structures used by programmers were programmed in assembly since the 1960s. At that time, applications were programmed using it is easy and practical when it fits your current application.

Learning to use a particular library (like STL) is not a main goal; even though the data structure ideas illustrated with the diagrams were used by programmers

3. Focusing on all the advantages and disadvantages of making one choice of data structure compared to another choice ("choice B") compared to "choice A".

2. Combining of data structures by "mix-and-match" combining of data structures definitions plus member functions.

1. After combining of data structures like those illustrated by the above structures taught in books and courses.

I. Some ultimate course objectives:
the oldest item in stock (FIFO). This is taught in accounting courses.

Now be based on the wholesale price paid last (LIFO). The wholesale price paid for
inventory item typically varies with when it was bought. Should the profit from a sale
selling a unit from an inventory. The wholesale price the business had paid to buy one
several accounting practices used by businesses. People to decide what profit results from

In other words, a FIFO (first-in, first-out) store. (LIFO and FIFO are two of

to buy admission to a movie.

Queue: Sequence in which insertion (at the „front“) and removal (at the „front“) occur at OPPOSITE ENDS. It is like a line of people waiting at a ticket counter

LIFO (last-in, first-out) store.

Stack: Sequence in which insertion and removal occur only at ONE END (the „top“). (We omit a review of stack, recursion, expression, tree relationships.)
queue is filled.

number of characters actually read, so the library can calculate how much the
function to read up to another block again. This function typically returns the
When the buffer becomes empty, the library will detect this and call the system
remove them when they are accessed by say getc one character at a time.

store the characters when they are read a block at a time, and to
in blocks of 256 or more characters. The inputStream library uses a buffer (i.e., a
is more efficient for the process to call a system function to read (disk) the data
from an input stream. However, except for hand-typed input from a terminal, it
The getc() member function inputStream removes and returns the earliest character

The most prevalent use of queues in computers worldwide is for buffers. A buffer
reduction operator% is useful here.

There are more or less elegant ways to program this; the C++

module length from it to make it "wrap around" to the beginning of the array.

If either position advanced beyond the end of the array, code will subtract the

they are equal, the queue is empty.

variables or pointer variables (indicate the rear and the front of the queue. If

manage an array as a circular buffer. Two position indicators (subscript

A new implementation idea for a queue (described in detail in the text) is to

implemented very much like those we covered for stacks.

dynamically allocated arrays. The member functions of queue classes are

implemented several different ways; with linked lists, static sized arrays, and

like stacks and other sequence or list type containers, queues can be
The details are given in the chapter on tree applications. Items in a heap-ordered balanced binary tree; and NOT store them fully sorted.

**IN FACT:** It is more efficient to implement a priority queue by storing the sorted items, if the elements were sorted in decreasing order of priority, the first element would be the one removed. This does NOT mean they must be greatest, among all the elements currently in the container.

Priority, the removal operation will remove an element whose priority is a numerical value or other kind of value from a heap. From a heap-ordered set (called its priority queue) is a container where every inserted element must be given with
Computer scientists love to reduce one data structure problem to another.

Item will always be the earliest inserted item that was not yet removed. NEGATIVE of the current time (or count of items). Then, the top priority.

To implement a queue, insert each item with its priority value given by the

most recently inserted item that was not yet removed. current time (or count of items). Then, the top priority item will always be the
current time (or count of items). Then, the top priority item will always be the

Either a stack or a queue can be implemented using a priority queue:
accumulate some elements.

which the number of arrivals exceeds the number served, the queue will

that the rate the "customers" are served. If there is a period of time during

We mentioned that the length of a queue would be zero if the arrival rate is less

a for-loop to simulate the second by second passage of time.

simulation. One variable current-second is incremented by 1 each time through

We covered the "car wash" simulation from Chapter 8. It is a time-step type of
\{ 
\text{do } \text{if event } E, \text{ -T }; \text{ with future times } T, \text{ and create new events } E \text{.....} \\
\}

\text{Event}: \text{simulate (PG 0) } \text{ .................} \\
\text{ }
\{ 
\text{Event}: \text{simulate (WPG) }; \text{ }
\text{time } = \text{ event.time}; \text{ }
// \text{in the priority queue} \\
\text{//event is the earliest} \\
(\text{while (event equals WPG.top()) }) \\
\text{Discrete event simulation}
("cheap, standard, and super") which take different amounts of time.

so several cars can be washed at the same time, and qualities of washes
implementation more complex car wash simulations. There might be multiple "stalls" With this priority queue driven discrete event simulation pattern, we can easily
priority.

events are stored in the priority queue, so the earliest events have greatest
more new events to be scheduled at various times in the future. Those new
When an event is selected and then simulated, the simulation may cause one or
occurs in the future. The priority queue selects the next event scheduled to occur.
Discrete event simulation relies on a priority queue to hold the events that will
Discrete simulation. Such simulations are often done in social or scientific research.