|  |  |  |
| --- | --- | --- |
| CARRERA:  Ingeniería en Software | GUÍA  No. 01 | TIEMPO ESTIMADO:  1h y 20 min. |
| ASIGNATURA:  Estructura de Datos  NRC: 4851 | FECHA DE ELABORACION: dd-mm-aa  SEMESTRE: Septiembre 2019 – Febrero 2020 | |
| TÍTULO:  Proyecto 1 | DOCENTE: Ing. Fernando Solis. MsC. | |

OBJETIVO

p.e. Utilizacion de backtracking en C.

INSTRUCCIONES

p.e.

1. Utilice como material principal, aquel indicado en clase por el docente.
2. Utilice información consultada en Internet y conocimiento adquirido en clase.

ACTIVIDADES

1. Ubicación de recursos

p.e.

1. Formar grupos de máximo 2 personas por computador
2. Instalar el IDE para C++.
3. Planteamiento del problema

p.e.

* Desarrollar un programa que aplique los conocimientos aprendidos en clase.
* El proyecto consiste en la implementación de backtraking en la ejecución de permutaciones.

Marco teórico

Vuelta atrás (*Backtracking*) es una estrategia para encontrar soluciones a problemas que satisfacen restricciones. El término "backtrack" fue acuñado por primera vez por el matemático estadounidense [D. H. Lehmer](https://es.wikipedia.org/wiki/Derrick_Henry_Lehmer) en la [década de 1950](https://es.wikipedia.org/wiki/Década_de_1950).

Enfoques

Los problemas que deben satisfacer un determinado tipo de restricciones son problemas completos, donde el orden de los elementos de la solución no importa. Estos problemas consisten en un conjunto (o lista) de variables a la que a cada una se le debe asignar un valor sujeto a las restricciones del problema. La técnica va creando todas las posibles combinaciones de elementos para obtener una solución. Su principal virtud es que en la mayoría de las implementaciones se puede evitar combinaciones, estableciendo funciones de acotación (o poda) reduciendo el [tiempo](https://es.wikipedia.org/w/index.php?title=Eficiencia_de_los_algoritmos&action=edit&redlink=1) de ejecución.

Diseño e implementación

Esencialmente, la idea es encontrar la mejor combinación posible en un momento determinado, por eso, se dice que este tipo de algoritmo es una [búsqueda en profundidad](https://es.wikipedia.org/wiki/Búsqueda_en_profundidad). Durante la búsqueda, si se encuentra una alternativa incorrecta, la búsqueda retrocede hasta el paso anterior y toma la siguiente alternativa. Cuando se han terminado las posibilidades, se vuelve a la elección anterior y se toma la siguiente opción (hijo [si nos referimos a un árbol]). Si no hay más alternativas la búsqueda falla. De esta manera, se crea un árbol implícito, en el que cada nodo es un estado de la solución (solución parcial en el caso de nodos interiores o solución total en el caso de los nodos hoja).

Normalmente, se suele implementar este tipo de algoritmos como un procedimiento [recursivo](https://es.wikipedia.org/wiki/Recursividad). Así, en cada llamada al procedimiento se toma una variable y se le asignan todos los valores posibles, llamando a su vez al procedimiento para cada uno de los nuevos estados. La diferencia con la [búsqueda en profundidad](https://es.wikipedia.org/wiki/Búsqueda_en_profundidad) es que se suelen diseñar funciones de cota, de forma que no se generen algunos estados si no van a conducir a ninguna solución, o a una solución peor de la que ya se tiene. De esta forma se ahorra espacio en memoria y tiempo de ejecución.

Heurísticas

Algunas heurísticas son comúnmente usadas para acelerar el proceso. Como las variables se pueden procesar en cualquier orden, generalmente es más eficiente intentar ser lo más restrictivo posible con las primeras (esto es, las primeras con menores valores posibles). Este proceso poda el [árbol de búsqueda](https://es.wikipedia.org/wiki/Árbol_(estructura_de_datos)) antes de que se tome la decisión y se llame a la subrutina recursiva.

Cuando se elige qué valor se va a asignar, muchas implementaciones hacen un examen hacia delante (FC, Forward Checking), para ver qué valor restringirá el menor número posible de valores, de forma que se anticipa en a) preservar una posible solución y b) hace que la solución encontrada no tenga restricciones destacadas.

Algunas implementaciones muy sofisticadas usan una función de cotas, que examina si es posible encontrar una solución a partir de una solución parcial. Además, se comprueba si la solución parcial que falla puede incrementar significativamente la eficiencia del algoritmo. Por el uso de estas funciones de cota, se debe ser muy minucioso en su implementación de forma que sean poco costosas computacionalmente hablando, ya que lo más normal es que se ejecuten en para cada nodo o paso del algoritmo. Cabe destacar, que las cotas eficaces se crean de forma parecida a las funciones [heurísticas](https://es.wikipedia.org/wiki/Heurística), esto es, relajando las restricciones para conseguir mayor eficiencia.

Con el objetivo de mantener la solución actual con coste mínimo, los algoritmos vuelta atrás mantienen el coste de la mejor solución en una variable que va variando con cada nueva mejor solución encontrada. Así, si una solución es peor que la que se acaba de encontrar, el algoritmo no actualizará la solución. De esta forma, devolverá siempre la mejor solución que haya encontrado

## En combinatoria

## La [combinatoria](https://es.wikipedia.org/wiki/Combinatoria) trata del número de diferentes maneras que existen de considerar conjuntos formados a partir de elementos de un conjunto dado, respetando ciertas reglas, como el tamaño, el orden, la repetición, la partición. Así un problema combinatorio consiste usualmente en establecer una regla sobre cómo deben ser las [agrupaciones](https://es.wikipedia.org/wiki/Combinaciones) y determinar cuántas existen que cumplan dicha regla. Básicamente, tres asuntos: permutaciones, combinaciones y variaciones.

## Un tipo importante de esas agrupaciones son las llamadas permutaciones. Dada una *n*-[tupla](https://es.wikipedia.org/wiki/Tupla) ordenada de los elementos de un conjunto, el número de permutaciones es el número de *n*-tuplas ordenadas posibles.

### Fórmula del número de permutaciones[[editar](https://es.wikipedia.org/w/index.php?title=Permutación&action=edit&section=3)]

Dado un conjunto finito {\displaystyle A\,\!} de {\displaystyle n\,\!} elementos, el número de todas sus permutaciones es igual a [factorial de n](https://es.wikipedia.org/wiki/Factorial):

{\displaystyle n!=n(n-1)(n-2)\cdots 1\,\!}Demostración: Dado que hay {\displaystyle n\,\!}formas de escoger el primer elemento y, una vez escogido este, sólo tenemos {\displaystyle (n-1)\,\!} formas de escoger el segundo elemento, y así sucesivamente, vemos que cuando llegamos al elemento k-ésimo sólo tenemos {\displaystyle [n-(k-1)]\,\!} posibles elementos para escoger, lo que nos lleva a que tenemos {\displaystyle n(n-1)(n-2)\cdots 2\cdot 1\,\!} formas de ordenar el conjunto, justamente lo que enunciamos anteriormente.

Ejemplo: sea el conjunto A={1,2,3} en este caso hay 6 permutaciones, en forma compacta: 123, 132, 213, 231, 312, 321. En álgebra, para estudiar los grupos simétricos se presentan entre paréntesis y en dos filas, en la primera siempre aparece 1 2 3.

Una variante de lo mismo, si se va a formar un comité que involucra presidente, tesorero y secretario, habiendo tres candidatos a, b, c ; cuando se elige por sorteo los cargos sucesivamente, hay seis posibilidades u ordenaciones: abc, acb, bac, bca, cab, cba.

Conclusiones

* Se evidencia la forma en la que funciona el backtraking en la programación.
* Y el manejo de las combinaciones.

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* Module: Funciones.h |
| \* Author: Jonathan |
| \* Modified: miércoles, 3 de noviembre de 2019 4:09:41 p. m. |
| \* Purpose: Declaration of the class Funciones |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
|  |
| #ifndef FUNCIONES\_H\_INCLUDED |
| #define FUNCIONES\_H\_INCLUDED |
|  |
| #include <iostream> |
| #include <stdio.h> |
| #include <string.h> |
| #include <stdlib.h> |
| #include <fstream> |
| #include <windows.h> |
|  |
| using namespace std; |
|  |
| class Cadena { |
| public: |
| fstream enter; |
| int n, contador; |
| char \*copy; |
|  |
| void mostrar(string str) { |
| n = str.size(); |
| copy = (char\*) malloc(10 \* sizeof (char\*)); |
| strcpy(copy, str.c\_str()); |
| enter << permute(copy, 0, n - 1); |
| enter << endl; |
| enter.close(); |
| } |
|  |
| char permute(char \*a, int l, int r) { |
| if (l == r) |
| enter << a << endl; |
| else { |
| for (int i = l; i <= r; i++) { |
| swap(\*(a + l), \*(a + i)); |
| permute(a, l + 1, r); |
| swap(\*(a + l), \*(a + i)); |
| } |
| } |
| } |
|  |
| Cadena(string n) { |
| enter.open("combinaciones.txt", fstream::out); //para leer in, para salir es out escribir |
| enter << "combinaciones posibles de " << n << endl; |
| contador = 0; |
| } |
|  |
| void intercambio ( int \*x, int \*y ) { |
| int tmp; |
| tmp = \*x; |
| \*x = \*y; |
| \*y = tmp; |
| } |
|  |
| void generarPdf() { |
| enter.open("combinaciones.txt", std::fstream::in); |
| if (enter.fail()) { |
| cout << "ERROR"; |
| } else { |
| enter.close(); |
| system("txt2pdf.exe combinaciones.txt com.pdf -oao -pfs60 -pps43 -ptc0 -width3000 -height2000"); |
| cout << "EXITO"; |
| } |
| } |
| }; |
|  |
| class Marquesina { |
| private: |
| short x, y; |
| const char\* mensaje; |
| public: |
| void gotoxy(short x, short y); |
| void moverMarquesina(const char\* mensaje); |
| }; |
|  |
| void Marquesina::gotoxy(short x, short y) { |
| COORD pos = {x, y}; |
| SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), pos); |
| } |
|  |
| void Marquesina::moverMarquesina(const char\* mensaje) { |
| int derecha = 0, n = 0, c = 1; |
| mensaje = "PERMUTACIONES MEDIANTE BACKTRACKING"; |
| do { |
| for (derecha = 1; derecha <= 75; derecha++) { |
| system("cls"); |
| gotoxy(derecha, 1); |
| cout << mensaje; |
| Sleep(10); |
| } |
| } while (c == 1); |
|  |
| } |
|  |
|  |

#endif // FUNCIONES\_H\_INCLUDED

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| \* Module: Principal (Main) |
| \* Author: Jonathan Picado |
| \* Author: Cesar Naula |
| \* Modified: miércoles, 5 de noviembre de 2019 20:09:41 p. m. |
| \* Purpose: Declaration of Main Function |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| #include "Funciones.h" |
| #include <conio.h> |
| #include <windows.h> |
| #define help 59 |
| using namespace std; |
|  |
| int main() { |
|  |
| string str; |
|  |
| cout << "Ingrese una Palabra " << endl; |
| cin >> str; |
| Cadena \*obj = new Cadena(str); |
| obj->mostrar(str); |
| obj->generarPdf(); |
| cout << "Se creo el archivo Combinaciones.txt" << endl; |
| return 0; |

}