Two ways to build the arch of hardware (explain) what is preferred and why and who use it?

Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC) are the two major approaches to processor architecture.

**Another**

Harvard Architecture (It is used in personal computers and small computers.)

von neumann architecture (It is used in micro controllers and signal processing.)

<https://www.geeksforgeeks.org/difference-between-von-neumann-and-harvard-architecture/>

Tabulation in recursion

#include <iostream>

#include <queue>

using namespace std;

#define ROW 5

#define COL 2

struct Person {

int age;

float height;

Person(int age, float height): age(age), height(height) { }};

struct CompareHeight {

bool operator()(Person const& p1, Person const& p2) {

// return "true" if "p1" is ordered

// before "p2", for example:

return p1.height < p2.height; }};

int main()

{

priority\_queue<Person, vector<Person>, CompareHeight> Q;

float arr[ROW][COL] = { { 30, 5.5 }, { 25, 5 },

{ 20, 6 }, { 33, 6.1 }, { 23, 5.6 } };

for (int i = 0; i < ROW; ++i) {

Q.push(Person(arr[i][0], arr[i][1]));

}

while (!Q.empty()) {

Person p = Q.top();

Q.pop();

cout << p.age << " " << p.height << "\n"; }

return 0;}

disadvantages of dynamic memory allocation

The problem with dynamic memory allocation is that it is not deallocated itself, developer responsibility to deallocate the allocated memory explicitly. If we cannot release the allocated memory, it can because of memory leak and make your machine slow. It is not only causes of memory leak but if you do not use the memory allocation properly, it can be the cause of the of memory fragmentation that is also serious issues.

There is one major problem with dynamic allocation, if you freed the memory before completed its task, then it can create hidden bug which is difficult to identify and can be a cause of the system crash or unpredictable value.

Overhead

Overallocation

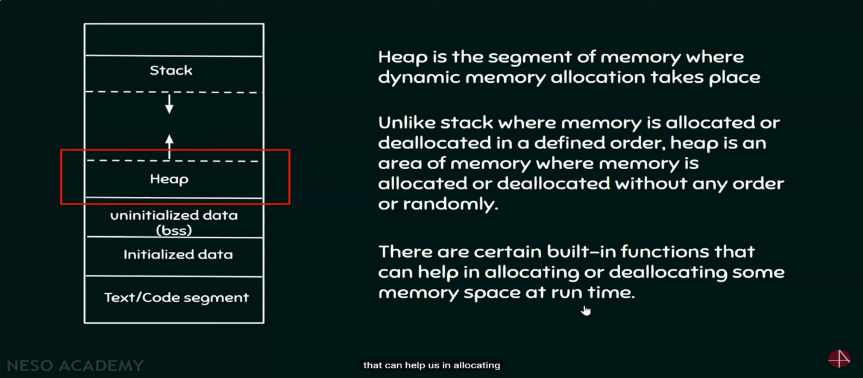
Difficult to access our bugs

slow

<https://aticleworld.com/problems-with-dynamic-memory-allocation/>

What difference between stack & heap (dynamic allocation)(all differences)

* Stack is a linear data structure whereas Heap is a hierarchical data structure.
* Stack memory will never become fragmented whereas Heap memory can become fragmented as blocks of memory are first allocated and then freed.
* Stack accesses local variables only while Heap allows you to access variables globally.
* Stack variables can’t be resized whereas Heap variables can be resized.
* Stack memory is allocated in a contiguous block whereas Heap memory is allocated in any random order.
* Stack doesn’t require to de-allocate variables whereas in Heap de-allocation is needed.
* Stack allocation and deallocation are done by compiler instructions whereas Heap allocation and deallocation is done by the programmer.

<https://www.guru99.com/stack-vs-heap.html#:~:text=Stack%20is%20a%20linear%20data,you%20to%20access%20variables%20globally>

C++ Frameworks and what is the best one?

* TensorFlow 146k most comman
* Caffe from Berkeley most pop
* Microsoft Cognitive Toolkit (CNTK)
* mlpack best syntax
* SHARK.
* Armadillo
* Faisis
* OpenNN
* FANN
* Boosting

The best alternative I've found is Shark, but as I said, it's still lacking and has only the more commonly used features, no LMA, annealing or PSO or anything of that level.

windows cronjob equivalent in c++

**croncpp – a C++ library for CRON expressions**

[**https://mariusbancila.ro/blog/2018/08/30/croncpp-a-cpp-library-for-cron-expressions/**](https://mariusbancila.ro/blog/2018/08/30/croncpp-a-cpp-library-for-cron-expressions/)

[**https://en.wikipedia.org/wiki/Cron**](https://en.wikipedia.org/wiki/Cron)

[**https://docs.oracle.com/cd/E12058\_01/doc/doc.1014/e12030/cron\_expressions.htm**](https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm)

[**https://github.com/mariusbancila/croncpp**](https://github.com/mariusbancila/croncpp)

What is pulling in cronjobs?

Working with cron pulls you out of the application—cron is a system level process. Not an application process. It’s challenging to give application developers access to anything at a system level. Developers shouldn’t care where their application runs… A good example of this is timezones. If a system person changes the timezone of a server, the cron may run at a different time than expected. The less the app developers have to worry about what they run on, the better.

How to update priority in queue?

removes the complexity of building, training, and deploying machine learning models at any scale.

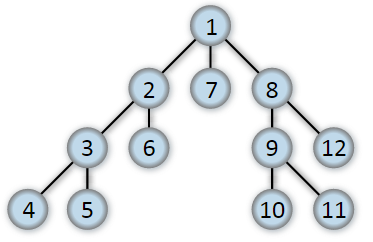
To actually update the queue, you either add an entry (which finds its appropriate slot), or remove an entry (either the highest priority one, or maybe another entry by some criteria like position or other key information like address/iterator, if the queue allows it).

To update the priority of an existing entry in a priority queue - can range from simple, to tricky, to impossible, depending on whether the priority information is mutable or not, and whether the queue can listen or respond to changing priority information.

The safest way is to remove the entry, change its priority, and re-add it to the queue.

DFS PFS algorithm

**Depth–first search (DFS) implemented in stack**

 The Depth–first search (DFS) algorithm starts at the root of the

tree (or some arbitrary node for a graph) and explored as far as

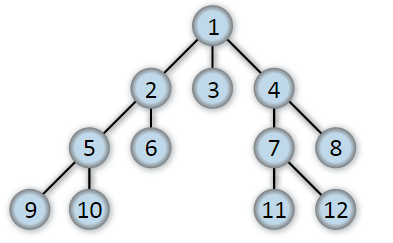
possible along each branch before backtracking.

**In depth- first there are three ways**

1. Root Left Right 🡪 preorder
2. Left root right 🡪 inorder
3. Left right Root🡪 postorder

**Breadth–first search (BFS) (use in queue)**

The Breadth–first search (BFS) algorithm also starts at the root of the tree (or some arbitrary node of a graph), but unlike DFS, it explores the neighbor nodes first, before moving to the next-level neighbors. In other words, BFS explores vertices in the order of their distance from the source vertex, where distance is the minimum length of a path from the source vertex to the node.



When to use DFS and BFS?

If we know the solution lies somewhere deep in a tree or far from the source vertex in the graph, use DFS. If we know the solution is not that far from the source vertex, use BFS.

If our tree is broad, use DFS as BFS will take too much memory. Similarly, if our tree is very deep, choose BFS over DFS.